08. Relational Database Design

[PM Jat, DAIICT, Gandhinagar]

Contents

Database Design as Process

Database design is a process of determining "good" database schema.

Measure of goodness depends on data model. At conceptual design, in ER, measure of goodness is subjective. It should capture complete and correct representation of database requirement.

In general, in addition to complete and correct representation, minimizing "data redundancies" is striving objective of database design. Data redundancy primarily means a fact/value is occurring at more than one place in the database.

> Relational Database Design as process
>
> Input:
>
> - Database Requirement Specifications
> - Data items and constraints
>
> Final Output:
>
> - (Good) Relational schema
> - Measure of Goodness: Minimum data redundancy

In relational design too, where outcome is good relational schema, minimizing data redundancy is objective. Relational model defines a formal method of measuring goodness of a relation, called Normal Forms - 1NF, 2NF, 3NF, BCNF, 4NF, 5NF.Higher the Normal Form, less the redundancies are, and "better" a relation is.

Before getting into details of these formal measures, let us first understand what the "data redundancy" is, and what are undesirable consequences of redundancy?

Database update Anomalies

Considering following set of attributes from XIT database – studid, name, stud_progid, cpi, pid, pname, intake, prog_did, did, dname for have three relations S(studid, name, stud_progid, cpi), P(pid, pname, intake, prog_did), D(did, dname). Hopefully it is "good" design (we will certify later, however). You should get the same schema, if we come through ER route (i.e. have ER model and derive relations using studied ER to relations mapping rules).

studid charact	name character	progid charact	cpi numeri
101	Rahul	BCS	8.70
102	Vikash	BEC	6.80
103	Shally	BEE	7.40
104	Alka	BEC	7.90
105	Ravi	BCS	9.30

pid charac	pname character vary	intake smallint	did charac
BCS	BTech(CS)	30	CS
BEC	BTech(ECE)	40	EE
BEE	BTech(EE)	40	EE
BME	BTech(ME)	40	ME

did chara	dname character varying(30)
CS	Computer Engineering
EE	Electrical Engineering
ME	Mechanical Engineering

Let us again take the question, "are above relations are good enough"? To answer this question, let us consider some of its alternatives. Compare and see why this is better over other options.

One possibility is, let us say, we merge Program and Department and have a single relation PD (is basically program relation with an additional attribute DName).

Below is combined relation, PD -

pid character	pname character vary	intake smallint	did chara	dname character varying(30)
BCS	BTech(CS)	30	CS	Computer Engineering
BEC	BTech(ECE)	40	EE	Electrical Engineering
BEE	BTech(EE)	40	EE	Electrical Engineering
BME	BTech(ME)	40	ME	Mechanical Engineering

Does above represent correct set of facts? Try understanding this -

- What will be interpretation of a tuple in combine relation PD?
- What will be the key in PD?
- How do we get tuple set for combine relation from individual relations - Program and Department?

Combined relation PD is basically natural join of both relations Program and Department.A tuple of PD contain all information of as program only, value of attribute dname is interpreted as name of department offering the program. Key of relation is PID. Do you see "data redundancy" here?

If not very obvious, let us also consider another case. Here we have combined relation of Student and Program relations (and let us call it SP), as shown below-

studid charact	name character	cpi numeric	progid charact	pname character vary	intake smallin	did chara
101	Rahul	8.70	BCS	BTech(CS)	30	CS
102	Vikash	6.80	BEC	BTech(ECE)	40	EE
103	Shally	7.40	BEE	BTech(EE)	40	EE
104	Alka	7.90	BEC	BTech(ECE)	40	EE
105	Ravi	9.30	BCS	BTech(CS)	30	CS

What are the undesirable consequences of redundancies?

Considering cost of modern storages, storage should not be of a concern, and even if it is, it probably gets compensated by reduced requirement performing joins wile answering queries (note most queries we need to join).

More serious problem associated with redundancies "database update anomalies".

One anomaly, you hopefully must have already noted, in combined PD, we cannot have a department details unless there is some program offered in that department (in PD). Or we cannot have a program detail unless there is some student attached with it (in SP)? This may be seriously undesirable.

1. Insertion Anomaly
2. Modify Anomaly
3. Deletion Anomaly

Insertion Anomaly

- Attempt inserting a new program in PD?
- Attempt inserting a new Department in PD?

While inserting, a new program, we need to supply all Department details as well. In this case we only have attribute DName; but could be more in other cases – for example in combined SP, attempt adding a new student? This is not only discomfort, but we may also end having inconsistent program details (in SP); inconsistent, since it is to be repeated with every program it offers.

Inserting a new Department in PD, without a program is not possible? We can choose to have null values for program portion of data but since PID is the key; it cannot be NULL.

These are the two examples of Insertion anomaly caused due to data redundancy.

Modification Anomaly

- Attempt changing program for a student in SP? Not only it require we supplying, all information of new program for the student in question, it may also lead database having inconsistent details of program in database; there is possibility of two tuple having same program information do not have same program information.

Deletion Anomaly

- If we delete program, and that program happens to be only program, department also gets deleted; that is we lose the department details as well!
- How do we delete a department? Deletion basically becomes changing department attributes value of a program to either NULL or some other department data. This is weird.

So, avoiding redundancy and hence avoiding update anomalies is the objective of relational schema design. As already said, normal forms are the measure of goodness of a relation schema. Normal forms are defined in terms of "functional dependencies". In section next we attempt understanding functional dependencies.

Function Dependencies

There are certain dependencies among attributes.

Consider all set of attributes in XIT schema, given below; search dname for a given prog-id, say BCS, you find that it gives you single value "Computer Engineering", even if it occurs at multiple places it gives us a single value. What does returned value indicates? It is name of the department that offers the program 'BCS'.

studid charact	name character	cpi numeri	progid charact	pname character varyi	intake smallint	did chara	dname character varying(30)
101	Rahul	8.70	BCS	BTech(CS)	30	CS	Computer Engineering
102	Vikash	6.80	BEC	BTech(ECE)	40	EE	Electrical Engineering
103	Shally	7.40	BEE	BTech(EE)	40	EE	Electrical Engineering
104	Alka	7.90	BEC	BTech(ECE)	40	EE	Electrical Engineering
105	Ravi	9.30	BCS	BTech(CS)	30	CS	Computer Engineering

On contrary, on searching of you student-id for given prog-id, you do not get a single value; and it is as expected that there are multiple students for a given prog-id. These observations are basically indicative of certain dependencies that occur between attributes in a database. Attribute dependencies come from problem domain and capture database constraints.

We call such dependencies as "Functional Dependencies"

Let us understand functional dependency from the concept of function in math; where you have

$$f: x \rightarrow y$$

Where let X and Y be attribute sets, and interpret it as - "for given the value x for attribute X, the function maps to a single value of attribute Y".If such a mapping exists then we say;X functionally determines Y, and express as $X \rightarrow Y$; then we also say that Y is functionally dependent on X.

It is important to be noted while studying functional dependencies, that, we see database as a single universal relation having all attributes of database. For instance, in XIT database, such a universal relation is RXIT(studid, name, stud_progid, cpi, prog_name, intake, prog_did, dname).

studid charact	name character	cpi numeri	progid charact	pname character varyi	intake smallint	did chara	dname character varying(30)
101	Rahul	8.70	BCS	BTech(CS)	30	CS	Computer Engineering
102	Vikash	6.80	BEC	BTech(ECE)	40	EE	Electrical Engineering
103	Shally	7.40	BEE	BTech(EE)	40	EE	Electrical Engineering
104	Alka	7.90	BEC	BTech(ECE)	40	EE	Electrical Engineering
105	Ravi	9.30	BCS	BTech(CS)	30	CS	Computer Engineering

Considering RXIT above, and see, if you find following functions true?

$f: studid \rightarrow name$

$f: studid \rightarrow intake$

Below is set of functional dependencies that hold true on XIT database.

studid→ name	**studid→ dname**
studid→ cpi	**progid→ pname**
studid→ progid	**progid→ did**
studid→ pname	**progid→ dname**
studid→ intake	**progid→ intake**
studid→ did	**did → dname**

Another perspective to understand function dependency

Let X and Y be some subset of attribute from R (may take RXIT mentioned above), and then we run following query on R-

```
SELECT distinct Y FROM R WHERE X = xval;
```

If the query returns a single value for Y, then we can say that there is a FD $X \rightarrow Y$. Try experimenting this with following relation, and verify above set of FDs.

What will be returned from following queries when executed on RXIT?

```
SELECT distinct dname FROM RXIT WHERE progid= 'BCS';
```

```
SELECT distinct intake FROM RXIT WHERE progid= 'BCS';
```

```
SELECT distinct studentid FROM RXIT WHERE progid= 'BCS';
```

And, we can infer that

$progid \rightarrow dname$

$progid \rightarrow intake$

$progid \not\rightarrow studentid$

Formal definition of Functional Dependency

Functional Dependencies are used to formally detect "redundancies in a relation". Below is a formal check that is used to specify functional dependencies in terms of repetition of data in tuples of a relation.

> If a tuple t1 in a relation r, has value x1 for attribute X and y1 for attribute Y; and if there is another tuple t2 that has x1 for X, and it also has y1 for Y (in t2).Then, we say that $X \rightarrow Y$, or X functionally determines Y; or Y is functionally dependent on X.
>
> In other words - for two tuples t1 and t2 form a relation r, if we have t1[X] = t2[X] and then we also have t1[Y] = t2[Y], then we say that $X \rightarrow Y$!

Note that FDs comes from meaning of attributes, not by looking at instances (in ainstance repetition may be merely coincidental).

Expression of Functional Dependencies

Remember in FD, $X \rightarrow Y$; both are sets. Suppose X is set of attribute A and B, and Y has attributes C and D, then X and Y, respectively represented as {A,B} and {C,D}, and function dependency is expressed as-

$$\{A, B\} \rightarrow \{C, D\}$$

However for short, often you find it expressed as $AB \rightarrow CD$

Exercise #1: Identify FDs in Company Database

Considering universal schema for company database,
RCOMP(ssn, name, salary, superssn, bdate, gender, emp_dno, dname, mgrssn, mgrstartdate, dlocation, pno, pname, plocation, hours, dep_name, dep_gender, dep_bdate, dep_relationship)

Does following expressions sound you correct in company schema? What check do we apply?

ssn→dname
ssn→ hours
ssn→ pname
superssn→ ssn
pno→ dname
(ssn, pno) → hours

Following is correct set of function dependencies in company database-

ssn→ name	ssn→ mgrstartdate	pno→ dname
ssn→ salary	dno→ dname	pno→ mgrssn
ssn→ superssn	dno→ mgrssn	pno→ mgrstartdate
ssn→ bdate	dno→ mgrstartdate	{ssn, pno} →hours
ssn→ gender	pno→ pname	{ssn, dep_name} →dep_gender
ssn→ dno	pno→ dno	{ssn, dep_name} → dep_bdate
ssn→ dname	pno→ plocation	{ssn, dep_name} → relationship
ssn→ mgrssn		

It is important to note that functional dependencies are drawn from problem domain and essentially capture database constraints. For example -

- when we say dno→ dname, it means is that there is only one name for a given dno. Given FD ssn→ superssn implies
- .36that there is only one supervisor for an employee.
- Being No FD superssn→ssn,implies that there can be multiple ssn for a given superssn; similary, Not being FD dno→dlocation, mean that there can be multiple locations for a department.
- Being FD {Course_No, AcadYr, Semester} → FacultyID implies that there can be only one instructor for a course offering.

Exercise #2: Function Dependencies for DA-Acad.

Following are all attributes; having understood of database requirements and various constraints. Attempt figuring out functional dependencies -

RACAD(StudetID, StdName, ProgID, Batch, CPI, Semester_SPI, Semester_CPI, CourseNo, CourseName, Credit, course_grade, FacultyID, FacultyName, AcadYear, Semester)

StudetID→ StdName StudetID→ ProgID StudetID→ Batch StudetID→ CPI CourseNo→ CourseName CourseNo→ Credit	FacultyID→ FacultyName {StudetID,AcadYear,Semester} → Semester_SPI {StudetID,AcadYear,Semester} → Semester_CPI {StudetID,CourseNo,AcadYear,Semester} → Course_Grade {CourseNo,AcadYear,Semester} → FacultyID

Key defined in terms of functional dependencies

Consider a relation R(X,Y), where X and Y are disjoint attribute Sets (and note X U Y is R). If we have FD X → Y, then X cannot repeat? Why? Because that will make Y to repeat and that will make tuple to repeat.

- Therefore X is super-key.
- If X is minimal, then it is Key.

Definition of Key in terms of FD: If a set of attributes X that functionally determines all other attributes of a relation R, and X is minimal, then X is Key.

Trivial Function Dependency

X → Y is trivial if Y is subset of X.This is trivial because subset will always be functionally determined by superset.For example: {SSN, NAME} → NAME. Proof?

Non-trivial FDs: FD X → Y is "non-trivial", when Y is not subset of X; but there can be some overlap of attributes; for example: {SSN} → {SSN, FNAME}.

A FD is said to be completely non-trivial if X and Y are disjoint in a FD, X →Y

Trivial FDs are simply discarded, as these are always true, and make no distinction.

Functional Dependency inference rules

In a given database scenario, there can be large number of functional dependencies identified. Some of them might be implied by other. A FD set is input to many of relational database design tasks (will see later). It is desirable that we have a minimal set of functional dependencies while not losing any functional dependencies. We call such set as "minimal FD set" or "base FD set". To have base set we need to figure out what FDs are implied from and what are base FD. Implied FDs are basically redundant FDs, and our minimal set may not have them.

For this, there are certain functional dependency inference rules that are used for computing implied or inferred functional dependencies.

Following are three basic inference rules, known as "Armstrong's axioms"-

- Reflexive Rule: if $Y \subseteq X$ then $X \rightarrow Y$; basically trivial FD rule.
  - Example: {SSN, PNO} → {PNO}
- Augmentation Rule: $\{X \rightarrow Y\} \models XZ \rightarrow YZ$
  - Example: {SSN} → {FNAME} ⊨{SSN, PPNO} → {FNAME, PNO}
- Transitive Rule: $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$
  - Example: {SSN} → {DNO} and {DNO} → {DNAME} ⊨ {SSN } → {DNAME}

We can prove these by going back to definition of function dependency!

There are certain derived rules from these basic axioms that become handy in identifying implied FDs; are following-

- Union Rule: $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$
  - Example: {SSN} → {FNAME} and {SSN} → {SALARY} ⊨ {SSN} → {FNAME, SALARY}
  - Proof:
    $X \rightarrow Y \models X \rightarrow XY$ (3)
    $X \rightarrow Z \models XY \rightarrow YZ$ (4)
    Transitively from (3) and (4) $\models X \rightarrow YZ$
- Decomposition Rule:$\{X \rightarrow YZ\} \models X \rightarrow Y$, and $X \rightarrow Z$
  - Example: {SSN} → {FNAME, SALARY} ⊨ {SSN} → {FNAME} and {SSN} → {SALARY}
  - Proof:
    $X \rightarrow YZ$ and $YZ \rightarrow Y \models X \rightarrow Y$
    $X \rightarrow YZ$ and $YZ \rightarrow Z \models X \rightarrow Z$

Exercise #3: Given FD set $\{AB \rightarrow C, BC \rightarrow AD, D \rightarrow E, CF \rightarrow B\}$, can we infer FD $AB \rightarrow D$?

$AB \rightarrow C$ (given) $\models AB \rightarrow BC$, and $BC \rightarrow AD \models BC \rightarrow D$

$AB \rightarrow BC$ and $BC \rightarrow D$ transitively $\models AB \rightarrow D$

Closure of Attributes

Closure of Attribute (or set of attributes) is set of all attributes that are functionally determined by the attribute(s). Closure is represented by X^+.

Here is algorithm for computing closure. It goes as following-

Begin with setting X to X^+, and scan all FDs, and wherever Left side of a FD is subset of X^+, add Right side attributes to X^+. This is repeated till we find X^+ is unchanged in an iteration.

Here is why algorithm will work correctly?

Consider a FD Y→Z, when we find left side of FD, i.e. Y, is subset of X^+, then we can say that
$X^+ \rightarrow Y$ (Reflexive Rule, trivial FD); we have following FDs, as premise:

$X \rightarrow X^+$; $X^+ \rightarrow Y$; and $Y \rightarrow Z$

Where from wee can infer that $X \rightarrow Z$, therefore Z can be appended to X^+.

```
Input: X, F
Output: X+

X+:= X;
repeat
oldX+ := X+
for each fd Y→Z in F do
if X+ is superset of Y then
        X+ := X+U Z;
until (X+= oldX+);
```

Exercise #4: Compute following attribute closure using this algorithm

1. XIT database: $\{StudentID\}^+$
2. Company Database: $\{SSN,PNO\}^+$
3. $F = \{AB \rightarrow C, BC \rightarrow AD, D \rightarrow E, CF \rightarrow B\}$, compute $\{AB\}^+$

Where can we use attribute closure?

- First, closure can be used to check if a functional dependency implied from given set F; for example, does FD $X \rightarrow Y$ implied from F? It can be answered by checking - if **Y is subset of X^+** then answer is YES, otherwise NO.
- Second, closure can be used for determining key of a relation.If closure of any set of attributes X contains all attributes of a relation, then we say that X is super-key of relation. If X is minimal then it is key. (will see little later)

Exercise #5: Given R(A,B,C,D,E,F), and $F = \{AB \rightarrow C, BC \rightarrow AD, D \rightarrow E, CF \rightarrow B\}$,
Does FD {A,B}→{D}inferred from F?

Apply attribute closure algorithm and, we get $\{A,B\}^+ = \{A,B,C,D,E\}$;

This implies that AB determines D.

Minimal FD set

A FD set, that does not have any implied FD called as Minimal FD set.

This has the advantage of having to work with less number of Functional Dependencies (still covers all FDs). For example, consider FDs in XIT database; following FDs are inferred and can be dropped. By dropping inferred FDs, we do not lose any FD/constraint.

studid→ pname

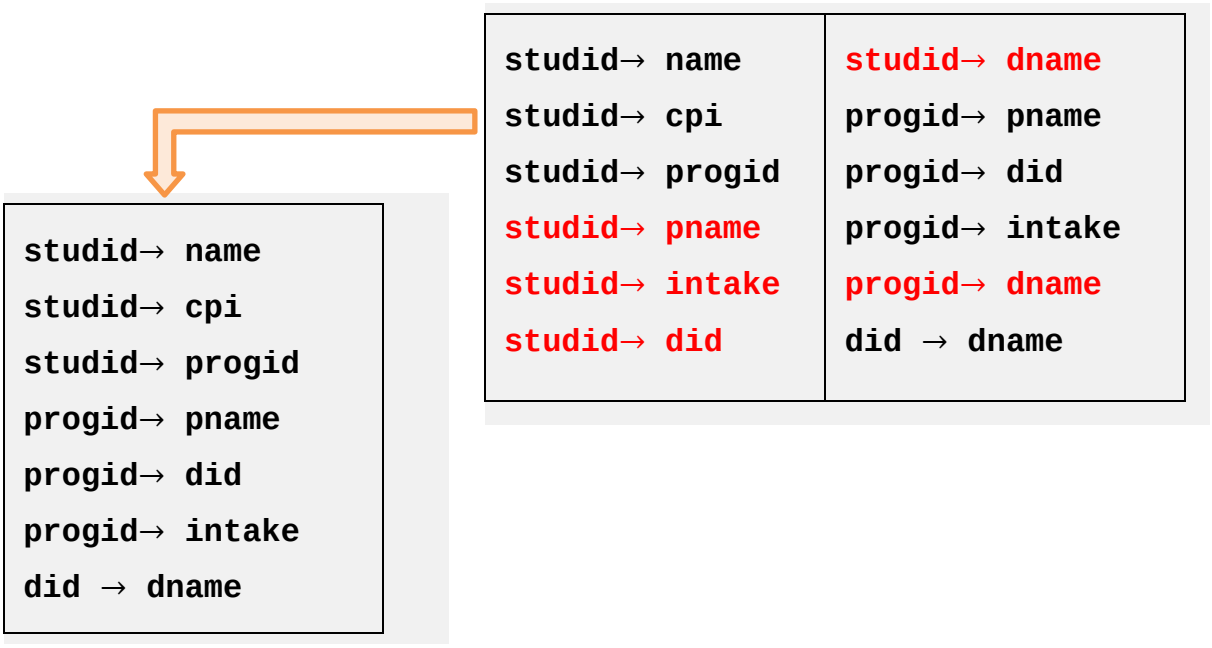
studid→ intake

studid→ did

studid→ dname

progid→ dname

Figure bellows shows minimal FD set for XIT database-



Algorithm for Determining Minimal Set

- Drop all trivial FDs
- Write the FDs in following (canonical) form-
  - Have only one attribute in right hand side and
  - Make left side "irreducible"
- Remove redundant FDs if any (i.e. No inferred FDs)
  - Should be easy to notice duplicate and trivial FD when written in canonical form
  - See if there are any transitively inferred FDs
  - See still there are some inferred FDs; remove them.

Korth's book calls minimal FD set "**Canonical Cover**" for a given FD set.

Computation of Key

For a relation R, and given set of FDs F, you can compute key for R as following-

- Pick one possible minimum set of attributes, X, and "compute closure", if closure includes all attributes of R, then X is key.
- Ensure X is minimal, if so, it is Key; X is minimal if closure of none of its subset contains all attributes of relation R.
- A relation might have multiple key, you should also see if there is some other attribute(s) Y, is a key.

Exercise #6: Compute Keys

1. R(ABCD), F = {AB → C, AC → D}
2. R(ABCDE), F = {AB →C, CD → E}
3. R(ABCDE), F = {A → B, C → D, AC → E}
4. R(A,B,C,D,E,F), F = {AB → C, BC → AD, D → E, CF → B}
5. R(CourseNo, Sem, AcadYear, InstructorID, StudentID, Grade).
   Identify FDs yourself.

Answersfor exercise #6:

1. AB
2. ABD
3. AC
4. ABF and CF
5. {Course-No, Semester, Acad-Year, Student-ID}

Exercises

1. Find out function dependency in attributes from TGMC scenario?
    a. MemberID
    b. MemberEmail
    c. TeamID
    d. TeamUserName
    e. TeamUserPassword
    f. MentorID
    g. MentorName
    h. InstituteID
    i. InstituteName

2. Do following FDs hold true? Attributes have been drawn from already seen situations.

    a. Does ISBN determine book title?
    b. Does book title determine ISBN?
    c. Does Accession No determine ISBN?
    d. Does ISBN determine Accession No?
    e. Does Author ID determine book title?
    Assuming that an author can write multiple books and a book can have multiple authors.

3. Compute minimal set for following FD sets

    a. $\{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
    b. $\{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$
    c. $\{AB \rightarrow C, B \rightarrow D, D \rightarrow A\}$
    d. $\{AB \rightarrow CD, B \rightarrow C, C \rightarrow D\}$
    e. $\{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
    f. $\{ABC \rightarrow D, A \rightarrow B\}$

4. Compute Keys for given relation R and FD set F-

    a. R (A, B, C, D, E, F), and following FD set
    $A \rightarrow BDE$
    $F \rightarrow A$
    b. R (A, B, C, D, E) and FD set
    $AB \rightarrow C$
    $CD \rightarrow E$
    $DE \rightarrow B$
    c. R (A, B, C, D, E, F, G) and FD set
    $A \rightarrow B, AC \rightarrow ED, B \rightarrow F, A \rightarrow G$

5. Given RMED(Trade-Name, Generic-Name, Manufacturer, Batch-No, Stock, MRP, Tax-Rate), and following FDs, Compute Key
    a. Trade-Name → Generic-Name
    b. Trade-Name → Manufacturer
    c. Batch-No → Trade-Name
    d. Batch-No → Stock
    e. Batch-No → MRP
    f. Generic-Name → Tax-Rate