SRM
Institute of Science and Technology

18CSC303J

Database Management System

Unit- IV

Relational Algebra

The relational algebra is a **procedural query language**.

- Consists of a **set of operations** that **take one or two relations as input** and **produce a new relation as their result.**

- The **fundamental operations** in the relational algebra are **select, project, union, set difference, Cartesian product, and rename.**

- In addition to the fundamental operations, **there are several other operations**—namely, **set intersection, natural join, and assignment.**

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Fundamental Operations

The select, project, and rename operations are called unary operations, because they operate on one relation.

The other three operations operate on pairs of relations and are, **called binary operations.**

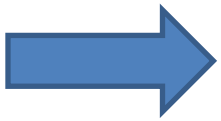
The Select Operation

- The select operation selects tuples that satisfy a given predicate.
- Lowercase Greek letter sigma (σ) is used to denote selection.
- The predicate appears as a subscript to σ.
- The argument relation is in parentheses after the σ.
- To select those tuples of the instructor relation where the instructor is in the “Physics” department, we write:

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

The instructor relation

$\sigma_{dept\_name = \text{“Physics”}}(instructor)$



ID	name	dept_name	salary
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Result of σ_{dept_name} =“Physics” (instructor)

Find all instructors with salary greater than $90,000.

$$\sigma_{salary>90000}\ (instructor)$$

ID	Name	Dept_Name	Salary
12121	Wu	Finance	90000
22222	Einstein	Physics	95000
83821	Brandt	Comp. Sci.	92000

The Select Operation

- In general, we allow comparisons using =, ≠, <, ≤, >, and ≥ in the selection predicate.

- We can combine several predicates into a larger predicate by using the connectives and (∧), or (∨), and not (¬).

- To find the instructors in Physics with a salary greater than $90,000

$$\sigma_{dept\_name = \text{“Physics”} \wedge salary > 90000}(instructor)$$

The Select Operation

- The selection predicate may include comparisons between two attributes.
- To illustrate, consider the relation department.
- To find all departments whose name is the same as their building name,

$$\sigma_{dept\_name = building}(department)$$

The Project Operation

- Suppose we want to list all instructors' ID, name, and salary, but do not care about the dept name.
- The project operation allows us to produce this relation.
- The project operation is a unary operation that returns its argument relation, with certain attributes left out.
- Since a relation is a set, any duplicate rows are eliminated.
- Projection is denoted by the uppercase Greek letter pi (Π).

The Project Operation

- We list those attributes that we wish to appear in the result as a subscript to Π.
- The argument relation follows in parentheses.
- We write the query to produce such a list

$$\Pi_{ID,\ name,\ salary}(instructor)$$

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

Composition of Relational Operations

- The fact that the result of a relational operation is itself a relation is important.

- Consider the more complicated query "Find the name of all instructors in the Physics department."

$$\Pi_{name}\ (\sigma_{dept\_name = \text{"Physics"}}\ (instructor))$$

- **Instead of giving the name of a relation as the argument of the projection operation,**
  - ***Give an expression that evaluates to a relation.***

Composition of Relational Operations

- **In general, the result of a relational-algebra operation** is ***of the same type (relation) as its inputs,***
  - ***relational-algebra operations can be composed together into a relational-algebra expression.***

- **Composing relational-algebra operations into relational-algebra expressions**
  - ***composing arithmetic operations(such as+,−, *, and÷) into arithmetic expressions.***

The Union Operation

- Consider a query to find the set of all courses taught in the Fall 2009 semester, the Spring 2010 semester, or both.

- The information is contained in the section relation as shown in figure.

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

The section relation

To find the set of all courses taught in the Fall 2009 semester

$$\Pi_{course\_id}\ (\sigma_{semester = \text{“Fall”} \wedge year=2009}\ (section))$$

To find the set of all courses taught in the Spring 2010 semester

$$\Pi_{course\_id}\ (\sigma_{semester = \text{“Spring”} \wedge year=2010}\ (section))$$

The Union Operation

- To answer the query, we need the union of these two sets; that is, we need all section IDs that appear in either or both of the two relations.

- We find these data by the binary operation union, denoted, as in set theory, by ∪.

- The expression needed is

$$\Pi_{course\_id}\ (\sigma_{semester = \text{“Fall”} \wedge year = 2009}\ (section)) \cup \Pi_{course\_id}\ (\sigma_{semester = \text{“Spring”} \wedge year = 2010}\ (section))$$

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

The Union Operation

- There are 8 tuples in the result, even though there are 3 distinct courses offered in the Fall 2009 semester and 6 distinct courses offered in the Spring 2010 semester.

- Since relations are sets, duplicate values such as CS-101, which is offered in both semesters, are replaced by a single occurrence.

- A union operation r ∪ s to be valid, we require that two conditions hold:

The Set-Difference Operation

- denoted by –
- allows us to find tuples that are in one relation but are not in another.
- The expression r – s produces a relation containing those tuples in r but not in s.

find all the courses taught in the Fall 2009 semester but not in Spring 2010 semester

$$\Pi_{course\_id}\,(\sigma_{semester = \text{“Fall”} \wedge year = 2009}\,(section)) - \Pi_{course\_id}\,(\sigma_{semester = \text{“Spring”} \wedge year = 2010}\,(section))$$

course_id
CS-347
PHY-101

The Set-Difference Operation

- must ensure that set differences are taken between compatible relations.

- Therefore, for a set-difference operation r − s to be valid, we require that the relations r and s be of the same arity, and that the domains of the i^{th} attribute of r and the i^{th} attribute of s be the same, for all i.

The Cartesian-Product Operation

- Denoted by a cross (×),
- Allows us to combine information from any two relations.
- We write the Cartesian product of relations r1 and r2 as r1 × r2.
- Recall that a relation is by definition a subset of a Cartesian product of a set of domains.

The Cartesian-Product Operation

- From that definition, we should already have an intuition about the definition of the Cartesian-product operation.

- However, since the same attribute name may appear in both r1 and r2, we need to devise a naming schema to distinguish between these attributes.

- We do so here by attaching to an attribute the name of the relation from which the attribute originally came.

- For example, the relation schema for r = instructor × teaches is

(instructor.ID, instructor.name, instructor.dept_name, instructor.salary
teaches.ID, teaches.course_id, teaches.sec_id, teaches.semester, teaches.year)

- With this schema, we can distinguish instructor.ID from teaches.ID.
- For those attributes that appear in only one of the two schemas, we shall usually drop the relation-name prefix.
- This simplification does not lead to any ambiguity.
- We can then write the relation schema for r as:

(instructor.ID, name, dept_name, salary
teaches.ID, course_id, sec_id, semester, year)

The Rename Operation

- Unlike relations in the database, the results of relational-algebra expressions do not have a name that we can use to refer to them.

- It is useful to be able to give them names; the rename operator, denoted by the lowercase Greek letter rho (ρ),

- Given a relational-algebra expression E, the expression

$$\rho_x (E)$$

- returns the result of expression E under the name x.

The Rename Operation

- A relation r by itself is considered a (trivial) relational-algebra expression.

- Apply the rename operation to a relation r to get the same relation under a new name.

- A second form of the rename operation is as follows: Assume that a relational algebra expression E has arity n. Then, the expression

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

- returns the result of expression E under the name x, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

Formal Definition of the Relational Algebra

- The operations allow us to give a complete definition of an expression in the relational algebra.

- A basic expression in the relational algebra consists of either one of the following:
  - ***A relation in the database***
  - ***A constant relation***
- A constant relation is written by listing its tuples within { },
- for example { (22222, Einstein, Physics, 95000), (76543, Singh, Finance, 80000) }.

Formal Definition of the Relational Algebra

- A general expression in the relational algebra is constructed out of smaller subexpressions.

- Let E1 and E2 be relational-algebra expressions.

- Then, the following are all relational-algebra expressions

  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_P(E_1)$, where P is a predicate on attributes in E_1
  - $\Pi_S(E_1)$, where S is a list consisting of some of the attributes in E_1
  - $\rho_x(E_1)$, where x is the new name for the result of E_1

The Set-Intersection Operation

- The first additional relational-algebra operation that we shall define is set intersection (∩).

Find the set of all courses taught in both the Fall 2009 and the Spring 2010 semesters.

$$\Pi_{course\_id}\ (\sigma_{semester = \text{“Fall”} \wedge year = 2009}\ (section)) \cap \Pi_{course\_id}\ (\sigma_{semester = \text{“Spring”} \wedge year = 2010}\ (section))$$

course_id
CS-101

Courses offered in both the Fall 2009 and Spring 2010 semesters

The Set-Intersection Operation

- Rewrite any **relational-algebra expression that uses set intersection**
- by replacing the intersection operation with a pair of set-difference operations

$$r \cap s = r - (r - s)$$

- Set intersection is not a fundamental operation and does not add any power to the relational algebra.

- It is simply more convenient to write r ∩ s than to write r – (r – s).

The Natural-Join Operation

- The natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.

- It is denoted by the join symbol ⋈

- The natural-join operation forms a Cartesian product of its two arguments,

- Performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes.

The Natural-Join Operation

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

The natural join of the instructor relation with the teaches relation

The Natural-Join Operation

- The result relation, has only 13 tuples, the ones that give information about an instructor and a course that that instructor actually teaches.

- We do not repeat those attributes that appear in the schemas of both relations; rather they appear only once.

- Notice **also the order in which the attributes are listed**
  - ***first the attributes common to the schemas of both relations,***
  - ***second those attributes unique to the schema of the first relation, and***
  - ***finally, those attributes unique to the schema of the second relation.***

The Natural-Join Operation

- The definition of natural join is complicated, the operation is easy to apply.

Find the names of all instructors together with the course id of all courses they taught.

$$\Pi_{name,\ course\_id}\ (instructor \bowtie teaches)$$

The Natural-Join Operation

- **The schemas for instructor and teaches have the attribute ID in common,**
  - ***the natural-join operation considers only pairs of tuples that have the same value on ID***.

- **It combines each such pair of tuples *into a single tuple on the union of the two schemas;*** that is, (ID, name, dept name, salary, course id).

- After performing the projection, we obtain the relation in Figure.

The Natural-Join Operation

- Consider two relation schemas R and S—which are, of course, lists of attribute names.

- If we consider the schemas to be sets, rather than lists,
  - ***Denote those attribute names that appear in both R and S by $R \cap S$,***
  - ***Denote those attribute names that appear in R, in S, or in both by $R \cup S$.***

- Similarly, those attribute names that appear in R but not S are denoted by $R - S$, whereas $S - R$ denotes those attribute names that appear in S but not in R.

The Natural-Join Operation

name	*course_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

Result of $\Pi_{name,\ course\_id}\ (instructor \bowtie teaches)$

Formal definition of the natural join

- Consider two relations r (R) and s(S).

- The natural join of r and s, denoted by r s, is a relation on schema R ∪ S formally defined as follows

$$r \bowtie s = \Pi_{R \cup S} (\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} (r \times s))$$

- where $R \cap S = \{A_1, A_2, \ldots, A_n\}$.

- Please note that if r (R) and s(S) are relations without any attributes in common,

$R \cap S = \emptyset$, then r s = r × s.

The Natural-Join Operation

Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach."

$$\Pi_{name,title}\left(\sigma_{dept\_name=\text{"Comp. Sci."}}\left(instructor \bowtie teaches \bowtie course\right)\right)$$

name	*title*
Brandt	Game Design
Brandt	Image Processing
Katz	Image Processing
Katz	Intro. to Computer Science
Srinivasan	Intro. to Computer Science
Srinivasan	Robotics
Srinivasan	Database System Concepts

Figure 6.16 Result of $\Pi_{name,title}\left(\sigma_{dept\_name=\text{"Comp. Sci."}}\left(instructor \bowtie teaches \bowtie course\right)\right)$.

The Natural-Join Operation

- we wrote instructor teaches course without inserting parentheses to specify the order in which the natural-join operations on the three relations should be executed.

$$(instructor \bowtie teaches) \bowtie course$$
$$instructor \bowtie (teaches \bowtie course)$$

- The natural join is associative

- The theta join operation is a variant of the natural-join operation that allows us to combine a selection and a Cartesian product into a single operation.

The Assignment Operation

- It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables.

- The assignment operation, denoted by ←, works like assignment in a programming language.

- To illustrate this operation, consider the definition of the natural-join operation.
- We could write r s as:

$$
\begin{aligned}
temp1 &\leftarrow R \times S \\
temp2 &\leftarrow \sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} (temp1) \\
result &= \Pi_{R \cup S} (temp2)
\end{aligned}
$$

The Assignment Operation

- The evaluation of an assignment does not result in any relation being displayed to the user.
- Rather, the result of the expression to the right of the ← is assigned to the relation variable on the left of the←.
- This relation variable may be used in subsequent expressions.
- With the assignment operation, a query can be written as a sequential program consisting of
  - ***a series of assignments***
  - ***followed by an expression***
  - ***whose value is displayed as the result of the query.***

The Assignment Operation

- For relational-algebra queries, assignment must always be made to a temporary relation variable.
- Assignments to permanent relations constitute a database modification.
- The assignment operation does not provide any additional power to the algebra.
- It is, a convenient way to express complex queries.

Outer join Operations

- An extension of the join operation to deal with missing information.
- Suppose that there is some instructor who teaches no courses.
- The tuple in the ***"instructor relation"*** for that particular instructor would not satisfy the condition of a natural join with the ***"teaches relation"***
- instructor's data would not appear in the result of the natural join, shown in Figure.

Outer join Operations

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

The instructor relation

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

The teaches relation

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

The natural join of the instructor relation with the teaches relation

Outer join Operations

- For example,
- instructors Califieri, Gold, and Singh do not appear in the result of the natural join, since they do not teach any course.

- More generally, some tuples in either or both of the relations being joined may be “lost” in this way.

- The outer join operation works in a manner similar to the natural join operation we have already studied, but preserves those tuples that would be lost in an join by creating tuples in the result containing null values.

Outer join Operations

- use the outer-join operation to avoid this loss of information.

- actually three forms of the operation:
  - ***left outer join, denoted*** ⟕,
  - ***right outer join, denoted ; and*** ⟖;
  - ***full outer join, denoted*** ⟗.

- All three forms of outer join compute the join, and add extra tuples to the result of the join.

- For example, the results of the expression instructor ⋈ teaches and teaches ⟖ instructor appear in Figures

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
33456	Gold	Physics	87000	*null*	*null*	*null*	*null*
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
58583	Califieri	History	62000	*null*	*null*	*null*	*null*
76543	Singh	Finance	80000	*null*	*null*	*null*	*null*
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Result of $instructor \sqsupset\!\bowtie teaches$.

ID	course_id	sec_id	semester	year	name	dept_name	salary
10101	CS-101	1	Fall	2009	Srinivasan	Comp. Sci.	65000
10101	CS-315	1	Spring	2010	Srinivasan	Comp. Sci.	65000
10101	CS-347	1	Fall	2009	Srinivasan	Comp. Sci.	65000
12121	FIN-201	1	Spring	2010	Wu	Finance	90000
15151	MU-199	1	Spring	2010	Mozart	Music	40000
22222	PHY-101	1	Fall	2009	Einstein	Physics	95000
32343	HIS-351	1	Spring	2010	El Said	History	60000
33456	*null*	*null*	*null*	*null*	Gold	Physics	87000
45565	CS-101	1	Spring	2010	Katz	Comp. Sci.	75000
45565	CS-319	1	Spring	2010	Katz	Comp. Sci.	75000
58583	*null*	*null*	*null*	*null*	Califieri	History	62000
76543	*null*	*null*	*null*	*null*	Singh	Finance	80000
76766	BIO-101	1	Summer	2009	Crick	Biology	72000
76766	BIO-301	1	Summer	2010	Crick	Biology	72000
83821	CS-190	1	Spring	2009	Brandt	Comp. Sci.	92000
83821	CS-190	2	Spring	2009	Brandt	Comp. Sci.	92000
83821	CS-319	2	Spring	2010	Brandt	Comp. Sci.	92000
98345	EE-181	1	Spring	2009	Kim	Elec. Eng.	80000

Result of $teaches \bowtie\!\sqsubset instructor$.

Left Outer join Operations

- The left outer join takes all tuples in the left relation
  - ***did not match with any tuple in the right relation,***
  - ***pads the tuples with null values for all other attributes from the right relation, and***
  - ***adds them to the result of the natural join.***

- In Figure, tuple (58583, Califieri, History, 62000, null, null, null, null), is such a tuple.

- All information from the left relation is present in the result of the left outer join.

Right Outer join Operations

- The right outer join is symmetric with the left outer join
- It pads tuples from the right relation that
  - ***did not match any from the left relation with nulls and***
  - ***adds them to the result of the natural join.***
- In Figure 6.18, tuple (58583, null, null, null, null, Califieri, History, 62000), is such a tuple.
- Thus, all information from the right relation is present in the result of the right outer join.

Full Outer join Operations

- The full outer join does both the left and right outer join operations,
  - ***padding tuples from the left relation that did not match any from the right relation,***
  - ***as well as tuples from the right relation that did not match any from the left relation, and***
  - ***adding them to the result of the join.***

Extended Relational-Algebra Operations

- Relational-algebra operations provide the ability to write queries that cannot be expressed using the basic relational-algebra operations.

- These operations are called extended relational-algebra operations.

Generalized Projection

- Extends the projection operation by allowing operations such as arithmetic and string functions to be used in the projection list.

- The generalized-projection operation has the form

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

- where
  - ***E is any relational-algebra expression,***
  - ***each of F_1, F_2, . . . , F_n is an arithmetic expression involving constants and attributes in the schema of E.***

Generalized Projection

- Generalized projection also permits operations on other data types, such as concatenation of strings.

$$\Pi_{ID, name, dept\_name, salary \div 12}(instructor)$$

Aggregation

- The second extended relational-algebra operation is the aggregate operation , $\mathcal{G}$

- Permits the use of aggregate functions such as min or average, on sets of values.

- Aggregate functions take a collection of values and return a single value as a result.
  - ***Sum, Average, Count, Min, Max***

The collections on which aggregate functions operate can have multiple occurrences of a value; the order in which the values appear is not relevant. *Such collections are called multisets*. Sets are a special case of multisets where there is only one copy of each element.

Find out the sum of salaries of all instructors

$$\mathcal{G}_{\mathbf{sum}(salary)}(instructor)$$

The result of the expression is a relation with a single attribute, containing a single row with a numerical value corresponding to the sum of the salaries of all instructors.

Aggregation

- There are cases where we **must eliminate multiple occurrences of a value** ***before computing an aggregate function.***

- **If we do want to eliminate duplicates,**
  - ***use the same function names***
  - ***with the addition of the hyphenated string "distinct" appended to the end of the function name***
  - ***for example, count-distinct***

Find the total number of instructors who teach a course in the Spring 2010 semester.

$$\mathcal{G}_{\text{count-distinct}(ID)}(\sigma_{semester=\text{“Spring”} \wedge year =2010}(teaches))$$

The aggregate function **“count-distinct”** ensures that even if an instructor teaches ore than one course, she is counted only once in the result.

To apply the aggregate function not to a single set of tuples, but instead to a group of sets of tuples.

Find the average salary in each department

$$_{dept\_name}\mathcal{G}_{\mathbf{average}(salary)}(instructor)$$

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Tuples of the instructor relation, grouped by the dept name attribute

dept_name	*salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

The result relation for the query “Find the average salary in each department”.

The Tuple Relational Calculus

The Tuple Relational Calculus

- When we write a relational-algebra expression, we provide a sequence of procedures that generates the answer to our query.

- The tuple relational calculus is a nonprocedural query language.

- It describes the desired information without giving a specific procedure for obtaining that information.

- A query in the tuple relational calculus is expressed as

$$\{t \mid P(t)\}$$

The Tuple Relational Calculus

- It is the **set of all tuples "t"** such that ***predicate "P" is true for "t".***

- Following our earlier notation, we use t[A] to denote the value of tuple "t" on attribute A, and we use t ∈ r to denote that tuple t is in relation r.

Find the ID, name, dept name, salary for instructors whose salary is greater than $80,000

$$\{t \mid t \in instructor \wedge t[salary] > 80000\}$$

Pitfalls in Relational Database Design

- Relational database design requires that **we find a "good" collection of relation schemas.**

- **A bad design may lead to**
  - ***Repetition of information.***
  - ***Inability to represent certain information.***

- Design Goals:
  - ***Avoid redundant data***
  - ***Ensure that relationships among attributes are represented***
  - ***Facilitate the checking of updates for violation of database integrity constraints***

Example

- Consider the relation schema:
- Lending-schema = (branch-name, branch-city, assets, customer-name, loan-number, amount)

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

Example

- Redundancy:
  - ***Data for branch-name, branch-city, assets are repeated for each loan that a branch makes***
  - ***Wastes space and complicates updating***

- Null values
  - ***Cannot store information about a branch if no loans exist***
  - ***Can use null values, but they are difficult to handle***

Decomposition

- Decompose the relation schema Lending-schema into:
  - ***Branch-schema = (branch-name, branch-city, assets)***
  - ***Loan-info-schema = (customer-name, loan-number, branch-name, amount)***

- All attributes of an original schema (R) must appear in the decomposition (R1, R2):

$$R = R1 \cup R2$$

- Lossless-join decomposition. For all possible relations r on schema R

$$r = \Pi R1\ (r) \bowtie R2\ (r)$$

Goal

- Decide whether a particular relation R is in "good" form.
- In the case that a relation R is not in "good" form,
  - ***decompose it into a set of relations {R_1, R_2, ..., R_n} such that each relation is in good form***
- The decomposition is a lossless-join decomposition
- Our theory is based on:
- functional dependencies
- multivalued dependencies

Decomposition

- The only way to avoid the repetition-of-information problem in the i*n_dep* schema is to decompose it into two schemas – instructor and *department* schemas.
- Not all decompositions are good. Suppose we decompose

  employee(ID, name, street, city, salary)

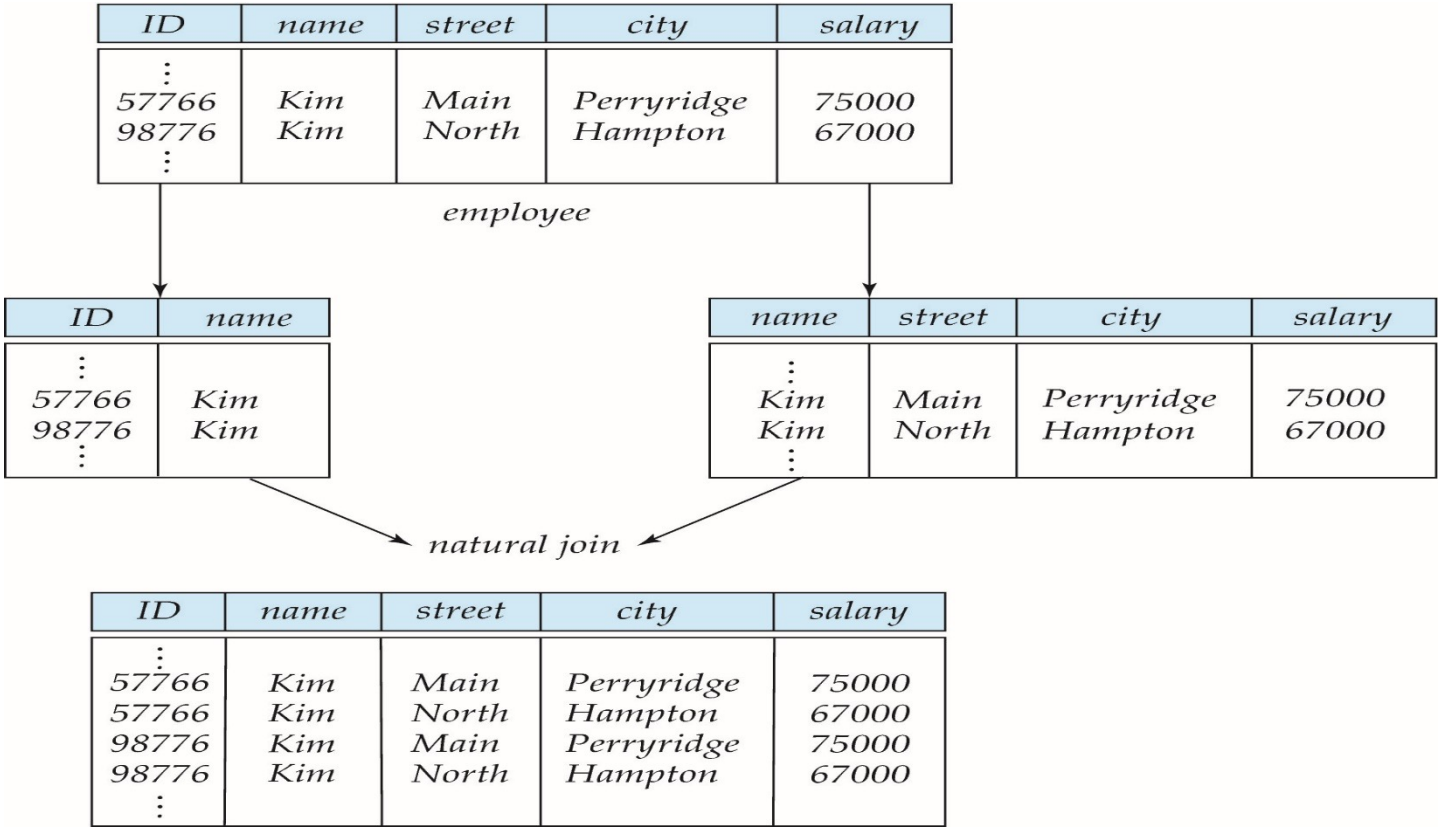
  into

  employee1 (*ID*, *name*)

  employee2 (*name*, *street, city, salary*)

  The problem arises when we have two employees with the same name
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**

A Lossy Decomposition



Lossless Decomposition

- Let *R* be a relation schema and let R_1 and R_2 form a decomposition of R . That is R = R_1 U R_2
- We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas R_1 U R_2
- Formally,

$$\prod_{R_1}(r) \quad \prod_{R_2}(r) = r$$

- And, conversely a decomposition is lossy if

$$r \subset \prod_{R_1}(r) \quad \prod_{R_2}(r) = r$$

Examples of Lossless Decomposition

- Decomposition of $R = (A, B, C)$
  $R_1 = (A, B)$ $R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$

A	B	C
α	1	A
β	2	B

Normalization theory

- Decide whether a particular relation *R* is in "good" form.
- In the case that a relation *R* is not in "good" form, decompose it into set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - Each relation is in good form
  - The decomposition is a lossless decomposition
- Our theory is based on:
  - Functional dependencies
  - Multivalued dependencies

Normalization

- The process of normalization is a formal method that identifies relational schemas based upon their primary or candidate keys and the functional dependencies that exists amongst their attributes.
- Normalization is primarily a tool to validate and improve a logical design so that it satisfies certain constraints that *avoid unnecessary duplication of data*.
- Normalization is the process of decomposing relation with anomalies to produce smaller, *well-structured* relations.
- Normalisation should remove redundancy, but not at the expense of data integrity.

- Transforming data from a problem into relations while ensuring data integrity and eliminating data redundancy.
  - Data integrity : consistent and satisfies data constraint rules
  - Data redundancy: if data can be found in two places in a single database (direct redundancy) or calculated using data from different parts of the database (indirect redundancy) then redundancy exists.
- Normalisation should remove redundancy, but not at the expense of data integrity.

First Normal form

- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account

- First normal form (1NF) deals with the `shape' of the record.
- A relation is in 1NF if, and only if, it contains no repeating attributes or groups of attributes.
- Example:
  - The Student table with the repeating group is not in 1NF
  - It has repeating groups, it is an `unnormalised table'.
- To remove the repeating group, either:
  - flatten the table and extend the key, or
  - decompose the relation, leading to First Normal Form

Flatten Table and Extend Primary Key

- The Student table with the repeating group can be written as:
  - Student(matric_no, name, date_of_birth, (subject, grade))
- If the repeating group was flattened, it would look something like:

  Student(matric_no, name, date_of_birth, subject, grade)
- This does not have repeating groups, but has redundancy. For every matric_no/subject combination, the student name and date of birth is replicated. This can lead to errors.
- Redundant data is the main cause of insertion, deletion, and updating anomalies.
  - Insertion anomaly – Subject is now in the primary key, we cannot add a student until they have at least one subject. Remember, no part of a primary key can be NULL.
  - Update anomaly – changing the name of a student means finding all rows of the database where that student exists and changing each one separately.
  - Deletion anomaly- for example deleting all database subject information also deletes student 960145.

Decomposing Relation

- The alternative approach is to split the table into two parts, one for the repeating groups and one of the non-repeating groups.
- The primary key for the original relation is included in both of the new relations

matric_no	subject	grade
960100	Databases	C
960100	Soft_Dev	A
960105	ISDE	D
960105	Soft_Dev	B
960105	ISDE	B
...	...	...
960150	Workshop	B

matric_no	name	date_of_birth
960100	Smith,J	14/11/1977
960105	White,A	10/05/1975
960120	Moore,T	21/08/1970
960145	Smith,J	09/01/1972
960150	Black,D	21/08/1973

Record: Student

- We now have two relations, Student and Record.
  - Student contains the original non-repeating groups
  - Record has the original repeating groups and the matric_no

Student(matric_no, name, date_of_birth)
Record(matric_no, subject, grade)

- This version of the relations does not have insertion, deletion, or update anomalies.
- Without repeating groups, we say the relations are in First Normal Form (1NF).

Second Normal Form

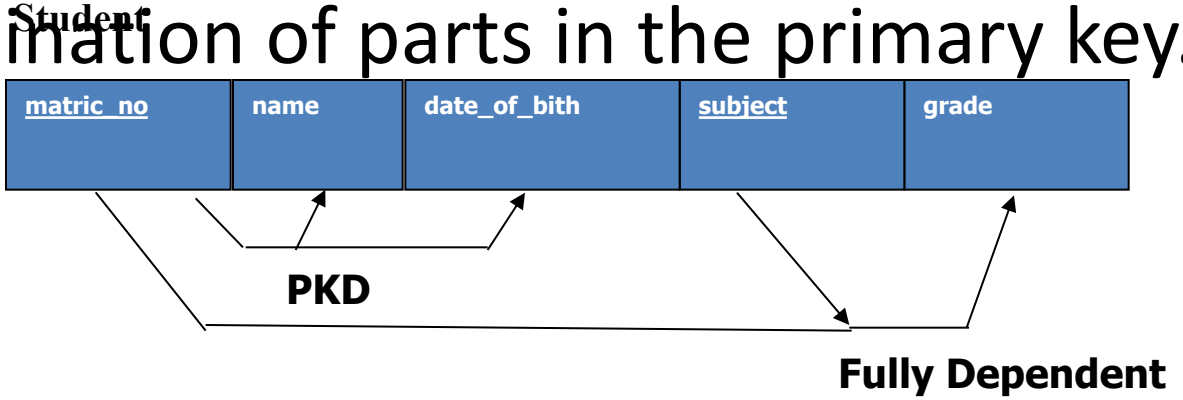
- A relation is in 2NF if, and only if, it is in 1NF and every non-key attribute is fully functionally dependent on the whole key.
- Thus the relation is in 1NF with no repeating groups, and all non-key attributes must depend on the whole key, not just some part of it. Another way of saying this is that there must be no partial key dependencies (PKDs).
- The problems arise when there is a compound key, e.g. the key to the Record relation - matric_no, subject. In this case it is possible for non-key attributes to depend on only part of the key - i.e. on only one of the two key attributes. This is what 2NF tries to prevent.

- Consider again the Student relation from the flattened table:

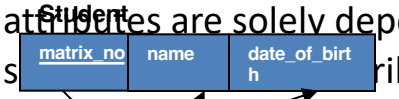
  Student(matric_no, name, date_of_birth, subject, grade)
- There are no repeating groups, so the relation is in 1NF
- However, we have a compound primary key - so we must check all of the non-key attributes against each part of the key to ensure they are functionally dependent on it.
  - matric_no determines name and date_of_birth, but not grade.
  - subject together with matric_no determines grade, but not name or date_of_birth.
- So there is a problem with potential redundancies

Dependency Diagram

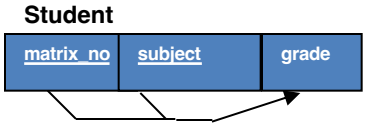
A dependency diagram is used to show how non-key attributes relate to each part or combination of parts in the primary key.



- This relation is not in 2NF
  - It appears to be two tables squashed into one.
  - the solutions is to split the relation into component parts.
- separate out all the attributes that are solely dependent on matric_no - put them in a new Student_details relation, with matric_no as the primary key
- separate out all the attributes that are solely dependent on subject - in this case no attributes are solely dependent on subject.
- s ributes that are solely dependent on matric_no + subject - put them into a separate Student relation, keyed on matric_no + subject



All attributes in each relation are fully functionally dependent upon its primary key



These relations are now in 2NF

Third Normal Form

- 3NF is an even stricter normal form and removes virtually all the redundant data :
- A relation is in 3NF if, and only if, it is in 2NF and there are no transitive functional dependencies
- Transitive functional dependencies arise:
  - when one non-key attribute is functionally dependent on another non-key attribute:
    - FD: non-key attribute -> non-key attribute
  - and when there is redundancy in the database
- By definition transitive functional dependency can only occur if there is more than one non-key field, so we can say that a relation in 2NF with zero or one non-key field must automatically be in 3NF

project_no	**manager**	**address**
p1	Black,B	32 High Street
p2	Smith,J	11 New Street
p3	Black,B	32 High Street
p4	Black,B	32 High Street

Project has more than one non-key field so we must check for transitive dependency:

- Address depends on the value of manager.
- From the table we can propose:

  Project(project_no, manager, address)
  manager -> address

- In this case address is transitively dependent on manager. The primary key is project_no, but the LHS and RHS have no reference to this key, yet both sides are present in the relation.
- Data redundancy arises from this
  - we duplicate address if a manager is in charge of more than one project
  - causes problems if we had to change the address- have to change several entries, and this could lead to errors.

- Eliminate transitive functional dependency by splitting the table
  - create two relations - one with the transitive dependency in it, and another for all of the remaining attributes.
  - split Project into Project and Manager.
- the determinant attribute becomes the primary key in the new relation - manager becomes the primary key to the Manager relation
- the original key is the primary key to the remaining non-transitive attributes - in this case, project_no remains the key to the new Projects table.

- Now we need to store the address only once
- If we need to know a manager's address we can look it up in the Manager relation
- The manager attribute is the link between the two tables, and in the Projects table it is now a foreign key.
- These relations are now in third normal form.

Project

project_no	manager
p1	Black,B
p2	Smith,J
p3	Black,B
p4	Black,B

Manager

manager	address
Black,B	32 High Street
Smith,J	11 New Street

BOYCE CODD NORMAL FORM

- Many practitioners argue that placing entities in 3NF is generally sufficient because it is rare that entities that are in 3NF are not also in 4NF and 5NF

  - The advanced forms of normalization are:
    - Boyce-Codd Normal Form
    - Fourth Normal Form
    - Fifth Normal Form

BOYCE CODD NORMAL FORM

- Boyce-Codd normal form (BCNF) is a more rigorous version of 3NF.
- BCNF is based on the concept of determinants. A determinant column is one on which some of the columns are fully functionally dependent.
- A relational table is in BCNF if and only if every determinant is a candidate key.

Boyce-Codd Normal Form (BCNF) Examples

Example 1 - Address (Not in BCNF)

Scheme → {City, Street, ZipCode }

1. Key1 → {City, Street }
2. Key2 → {ZipCode, Street}
3. No non-key attribute hence 3NF
4. {City, Street} → {ZipCode}
5. {ZipCode} → {City}
6. Dependency between attributes belonging to a key

Boyce Codd Normal Form (BCNF)

Example 2 - Movie (Not in BCNF)

Scheme → {MovieTitle, MovieID, PersonName, Role, Payment }

1. Key1 → {MovieTitle, PersonName}
2. Key2 → {MovieID, PersonName}
3. Both role and payment functionally depend on both candidate keys thus 3NF
4. {MovieID} → {MovieTitle}
5. Dependency between MovieID & MovieTitle Violates BCNF

Example 3 - Consulting (Not in BCNF)

Scheme → {Client, Problem, Consultant}

1. Key1 → {Client, Problem}
2. Key2 → {Client, Consultant}
3. No non-key attribute hence 3NF
4. {Client, Problem} → {Consultant}
5. {Client, Consultant} → {Problem}
6. Dependency between attributess belonging to keys violates BCNF

BCNF - Decomposition

1. Place the two candidate primary keys in separate entities
2. Place each of the remaining data items in one of the resulting entities according to its dependency on the primary key.

Example 1 (Convert to BCNF)

Old Scheme → {City, Street, ZipCode }

New Scheme1 → {ZipCode, Street}

New Scheme2 → {City, Street}

- Loss of relation {ZipCode} → {City}

Alternate New Scheme1 → {ZipCode, Street }

Alternate New Scheme2 → {ZipCode, City}

Decomposition – Loss of Information

1. If decomposition does not cause any loss of information it is called a **lossless** decomposition.
2. If a decomposition does not cause any dependencies to be lost it is called a **dependency-preserving** decomposition.
3. Any table scheme can be decomposed in a lossless way into a collection of smaller schemas that are in BCNF form. However the dependency preservation is not guaranteed.
4. Any table can be decomposed in a lossless way into 3rd normal form that also preserves the dependencies.
   - 3NF may be better than BCNF in some cases

Use your own judgment when decomposing schemas

BCNF - Decomposition

Example 2 (Convert to BCNF)

Old Scheme → {MovieTitle, MovieID, PersonName, Role, Payment }

New Scheme → {MovieID, PersonName, Role, Payment}

New Scheme → {MovieTitle, PersonName}

- Loss of relation {MovieID} → {MovieTitle}

New Scheme → {MovieID, PersonName, Role, Payment}

New Scheme → {MovieID, MovieTitle}

- We got the {MovieID} → {MovieTitle} relationship back

Example 3 (Convert to BCNF)

Old Scheme → {Client, Problem, Consultant}

New Scheme → {Client, Consultant}

New Scheme → {Client, Problem}

Fourth Normal Form (4NF)

- Fourth normal form eliminates independent many-to-one relationships between columns.
- To be in Fourth Normal Form,
  - a relation must first be in Boyce-Codd Normal Form.
  - a given relation may not contain more than one multi-valued attribute.

Example (Not in 4NF)

Scheme → {MovieName, ScreeningCity, Genre)

Primary Key: {MovieName, ScreeningCity, Genre)

1. All columns are a part of the only candidate key, hence BCNF
2. Many Movies can have the same Genre
3. Many Cities can have the same movie
4. Violates 4NF

Movie	ScreeningCity	Genre
Hard Code	Los Angles	Comedy
Hard Code	New York	Comedy
Bill Durham	Santa Cruz	Drama
Bill Durham	Durham	Drama
The Code Warrier	New York	Horror

Fourth Normal Form (4NF)

Example 2 (Not in 4NF)

Manager	Child	Employee
Jim	Beth	Alice
Mary	Bob	Jane
Mary	NULL	Adam

Scheme → {Manager, Child, Employee}

1. Primary Key → {Manager, Child, Employee}
2. Each manager can have more than one child
3. Each manager can supervise more than one employee
4. 4NF Violated

Example 3 (Not in 4NF)

Scheme → {Employee, Skill, ForeignLanguage}

1. Primary Key → {Employee, Skill, Language }
2. Each employee can speak multiple languages
3. Each employee can have multiple skills
4. Thus violates 4NF

Employee	Skill	Language
1234	Cooking	French
1234	Cooking	German
1453	Carpentry	Spanish
1453	Cooking	Spanish
2345	Cooking	Spanish

4NF - Decomposition

1. Move the two multi-valued relations to separate tables
2. Identify a primary key for each of the new entity.

Example 1 (Convert to 3NF)

Old Scheme → {MovieName, ScreeningCity, Genre}

New Scheme → {MovieName, ScreeningCity}

New Scheme → {MovieName, Genre}

Movie	Genre
Hard Code	Comedy
Bill Durham	Drama
The Code Warrier	Horror

Movie	ScreeningCity
Hard Code	Los Angles
Hard Code	New York
Bill Durham	Santa Cruz
Bill Durham	Durham
The Code Warrier	New York

4NF - Decomposition

Example 2 (Convert to 4NF)

Old Scheme → {Manager, Child, Employee}

New Scheme → {Manager, Child}

New Scheme → {Manager, Employee}

Manager	Child
Jim	Beth
Mary	Bob

Manager	Employee
Jim	Alice
Mary	Jane
Mary	Adam

Example 3 (Convert to 4NF)

Old Scheme → {Employee, Skill, ForeignLanguage}

New Scheme → {Employee, Skill}

New Scheme → {Employee, ForeignLanguage}

Employee	Skill
1234	Cooking
1453	Carpentry
1453	Cooking
2345	Cooking

Employee	Language
1234	French
1234	German
1453	Spanish
2345	Spanish

Multivalued Dependencies and Fourth Normal Form

Definition:

- A **multivalued dependency** (**MVD**) X —>> Y specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties, where we use Z to denote (R 2 (X ∪ Y)):
  - $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.
  - $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.
  - $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.
- An MVD X —>> Y in R is called a **trivial MVD** if (a) Y is a subset of X, or (b) X ∪ Y = R.

Multivalued Dependencies and Fourth Normal Form

Definition:

A relation schema *R* is in **4NF** with respect to a set of dependencies *F* (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency $X \twoheadrightarrow Y$ in F^+, *X* is a superkey for R.

- Note: F^+ is the (complete) set of all dependencies (functional or multivalued) that will hold in every relation state *r* of *R* that satisfies *F*. It is also called the **closure** of *F*.

Join Dependency

What is join dependency?

- If a table can be recreated by joining multiple tables and each of this table have a subset of the attributes of the table, then the table is in Join Dependency.
- It is a generalization of Multivalued Dependency
- If the join of R1 and R2 over C is equal to relation R, then we can say that a join dependency (JD) exists.
- Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.
- A JD ⋈ {R1, R2,..., Rn} is said to hold over a relation R if R1, R2,....., Rn is a lossless-join decomposition.
- The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R.
- Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Example: <employee>

Empname	EmpSkills	EmpJob (Assigned Work)
Tom	Networking	EJ001
Harry	Web Development	EJ002
Katie	Programming	EJ002

The table can be decomposed into 3 tables

<EmployeeSkills>

Emp Name	EmpSkills
Tom	Networking
Harry	Web Development
Katie	Programming

<EmployeeJob>

EmpName	EmpJob
Tom	EJ001
Harry	EJ002
Katie	EJ002

<JobSkills>

EmpSkills	EmpJob
Networking	EJ001
Web Development	EJ002
Programming	EJ002

Join dependency
{(EmpName, EmpSkills), (EmpName, EmpJob), (EmpSkills, EmpJob)}

Fifth Normal Form-5NF

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).
- Example:

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

5NF- Continued

- In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.
- Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.
- So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3.

5NF-Continued

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMESTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen