CH-1 RELATIONAL DATA MODEL

RELATIONAL MODEL CONCEPTS:

The relational model represents the database as a collection of relations. A relation is a table of values where each row represents a collection of data values.

Eg: The column names—Name, Student number, Class, and Major in STUDENT entity — specify how to interpret the data values in each row, based on the column each value is in. All values in a column are of the same data type

In the formal relational model terminology, a row is called a *tuple*, a column header is called an *attribute*, and the table is called a *relation.* The data type describing the types of values that can appear in each column is represented by a *domain* of possible values.

Domain: A domain D is a set of atomic values. Each value in the domain is indivisible. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.

Example: Set of telephone numbers- Numbers

Set of student names- Character strings representing names of students

They are logical definitions of domains. A data type or format is specified for each domain.

Relation Schema: A relation schema R(A1,A2,….,An) is made up of a relation R and a list of attributes A1,A2,….,An. Each attribute Ai is the name of the role played by some domain D in the relation schema R. D is called the domain of Ai and is denoted by dom(Ai). A relation schema is used to describe a relation.

The degree of a relation is the number of attributes in the relation schema. A relation or a relation state r of a relation schema R(A1,A2,….,An), is also denoted by r(R) is a set of n-tuples r={t1,t2,…,tn}. Each n-tuple is an ordered list of values t=<v1,v2,….,vn> where each value vi is 1<=i<=n is an element of dom(Ai) or a special null value.

Tuple: Each row in a relation is called a tuple.

Attribute: The column header in a relation is called an attribute of a relation.

CHARACTERISTICS OF RELATIONS

Ordering of tuples in a relation r(R): A relation is defined as a set of tuples .The tuples are *not* considered to be ordered, even though they appear to be in the tabular form.

Ordering of attributes in a relation schema R: We will consider the attributes in $R(A_1, A_2, ..., A_n)$ and the values in $t=<v_1, v_2, ..., v_n>$ to be *ordered* .

Values in a tuple: All values are considered *atomic* (indivisible). A special null value is used to represent values that are unknown or inapplicable to certain tuples

Notation: We refer to component values of a tuple t by $t[A_i] = v_i$ (the value of attribute A_i for tuple t).Similarly, $t[A_u, A_v, ..., A_w]$ refers to the subtuple of t containing the values of attributes $A_u, A_v, ..., A_w$,respectively

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21

RELATIONAL CONSTRAINTS: Constraints or restrictions imposed on the actual values

This can be classified into 3 categories

1. Model based constraints
2. Schema based constraints
3. Application based constraints

Schema based constraints

i. Domain constraints
ii. Key constraints
iii. Constraints on null
iv. Entity integrity constraints
v. Referential integrity constraints

I. **Domain constraints**: It specify the value of each attribute in the tuple which must be an atomic value .it is specified while creating the table itself .

II. **Key constraints** : These constraint defines the restriction on a particular attribute (column) or set of attributes

a) Super key: A set of attributes SK of R such that no two tuples *in any valid relation instance r(R)* will have the same value for SK.That is, for any distinct tuples t1 and t2 in r(R), t1[SK]≠t2[SK].

(Subset of attributes which give unique tuples i.e no distinct tuples can have the same values)

Eg:(USN,NAME,RNO,CLASS),(NAME,RNO,CLASS),(USN,NAME),

(RNO,CLASS),(NAME,RNO,PER)

Note: Set of all attributes always form a super key

b) Key: A minimal subset of super key which is still a superkey is called a key

Example: The CAR relation schema:

CAR(State, Reg#, SerialNo, Make, Model, Year)

Key1 = {State, Reg#},

Key2 = {SerialNo}, which are also superkeys.

{SerialNo, Make} is a superkey but *not* a key.

Figure 7.4 The CAR relation with two candidate keys: LicenseNumber and EngineSerialNumber.

CAR	LicenseNumber	EngineSerialNumber	Make	Model	Year
	Texas ABC-739	A69352	Ford	Mustang	96
	Florida TVP-347	B43696	Oldsmobile	Cutlass	99
	New York MPO-22	X83554	Oldsmobile	Delta	95
	California 432-TFY	C43742	Mercedes	190-D	93
	California RSK-629	Y82935	Toyota	Camry	98
	Texas RSK-629	U028365	Jaguar	XJS	98

c) Candidate keys: A relation schema may have more than one key,each of the keys is called a candidate key

d) **Primary key:** one of the candidate key is chosen arbitrary & is designated as primary key (An attribute which has unique values for each tuple,it is not null & used to recognized to each Tuple)

e) Foreign key : An attribute in a relation is foreign key if it is primary key of another relation

III) Null constraints :specify whether an attribute should have a value also.

RELATIONAL DATABASES SCHEMAS

Relational Database Schema: A set S of relation schemas that belong to the same database. S is the *name* of the database. $S = \{R_1, R_2, ..., R_n\}$

Figure 7.5 Schema diagram for the COMPANY relational database schema; the primary keys are underlined.

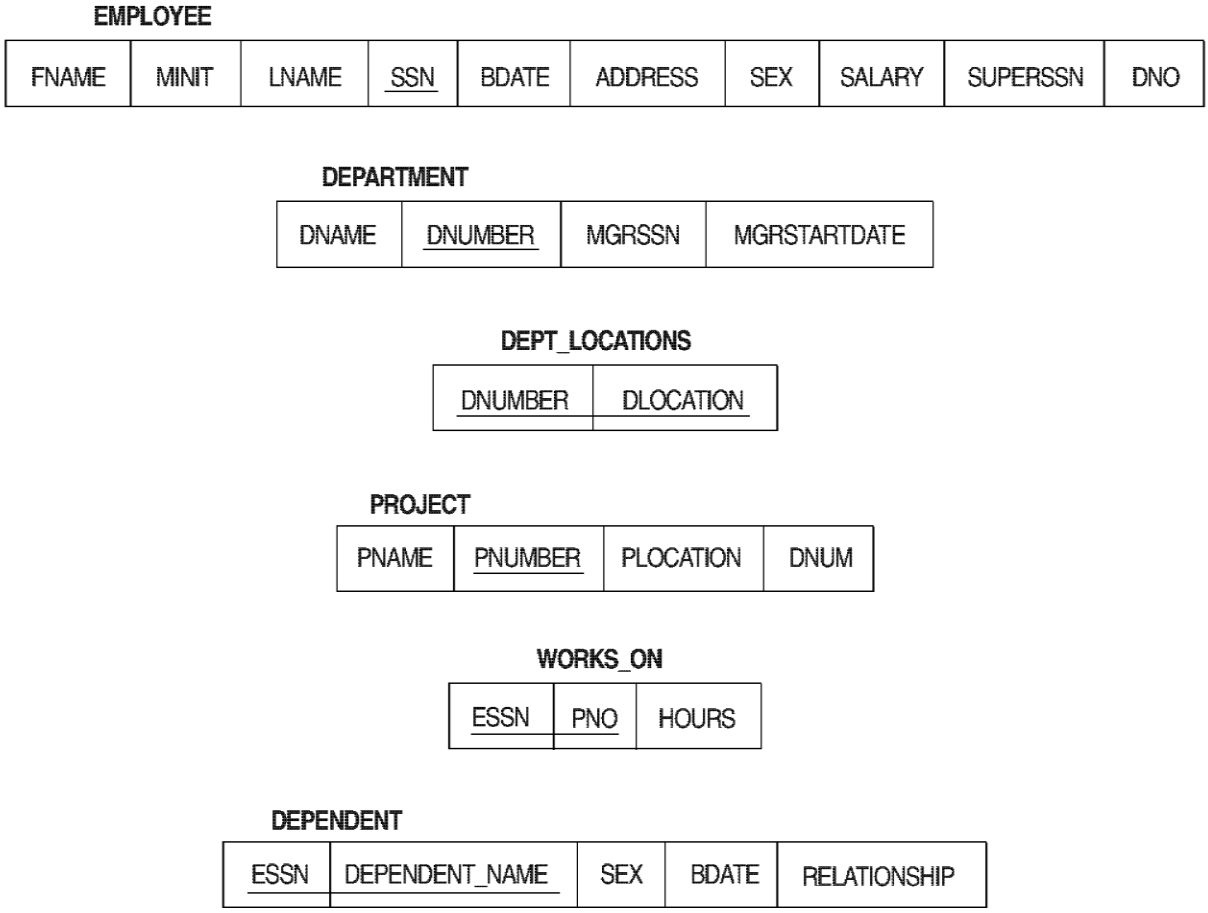


Figure 7.6 One possible relational database state corresponding to the COMPANY schema.

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
		Houston
		Stafford
		Bellaire
		Sugarland

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

4. Entity integrity constraints:it states that every relation has a primary key which are unique,not null The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R). This is because primary key values are used to *identify* the individual tuples. t[PK] ≠ null for any tuple t in r(R)

Note: Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.

5)Referential integrity constraints: it maintains common value among the rows of two relation.

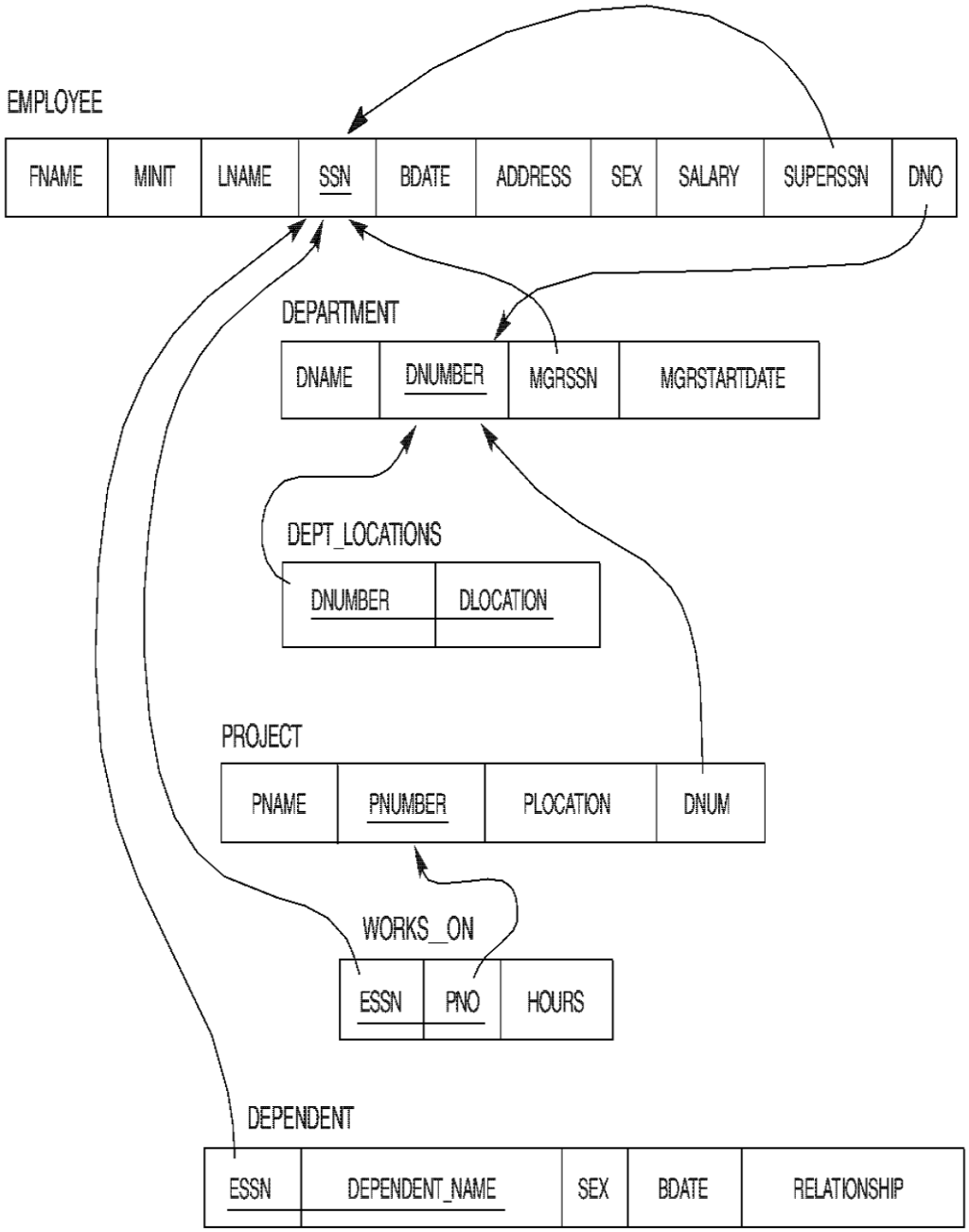
- A constraint involving *two* relations (the previous constraints involve a *single* relation).
- Used to specify a *relationship* among tuples in two relations: the referencing relation and the referenced relation.
- Tuples in the *referencing relation* R_1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the *referenced relation* R_2. A tuple t_1 in R_1 is said to reference a tuple t_2 in R_2 if $t_1[FK] = t_2[PK]$.
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R_1.FK to R_2.

Statement of the constraint

The value in the foreign key column (or columns) FK of the the referencing relation R_1 can be either:

1. A value of an existing primary key value of the corresponding primary key PK in the referenced relation $R_{2,}$, 2.
2. A value of fk in a tuple t1, of the current state r1(R1) either occurs as a value of pk for some tuple t2 in the current state r2(R2) or is nulla null.In case (2), the FK in R_1 should no be a part of its own primary key.

Figure 7.7 Referential integrity constraints displayed on the COMPANY relational database schema diagram.



Other Types of Constraints

I) State Constraints

II) Transition Constraints

I) State Constraints : Defines the constraints that a valid state of the database must satisfy

Semantic Integrity Constraints: Based on application semantics and cannot be expressed by the model on large class E.g: "the max. no. of hours per employee for all projects he or she works on is 56 hrs per week"

- A *constraint specification language is used to specify & enforce these constraints*
- *Mechanisms called* SQL-99 allows triggers and ASSERTIONS are used

3. **Functional dependency constraint**: which establishes a functional relationship among two sets of attributes X and Y.This constraints specifies that the value of X determines the value of Y in all states of a relation. It is denoted by X Y

II) Transition Constraints: It can be defined to deal with state changes in the database

Eg: the salary of an employee can only increase

UPDATE OPERATIONS, TRANSACTIONS,AND DEALING WITH CONSTRAINT VIOLATIONS

The operations of the relational model can be categorized into *retrievals* and *updates.* The relational algebra operations, which can be used to specify **retrievals**,

A relational algebra expression forms a new relation after applying a number of algebraic operators to an existing set of relations; its main use is for querying a database to retrieve information.

There are three basic operations that can change the states of relations in the database:

Insert, Delete, and Update (or Modify).

1) Insert is used to insert one or more new tuples in a relation,

2) Delete is used to delete tuples, and

3) Update (or **Modify**) is used to change the values of some attributes in existing tuples.

1) Insert operation: It provides a list of attribute values for a new tuple that is to be inserted into R.

Insert operation can violate the following constraints:

- Domain constraint can be violated if an attribute value does not match the specified domain.
- Key constraint can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R).
- Entity integrity constraint can be violated if the primary key of the new tuple t is null
- Referential integrity constraint can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation.

Example:

1) Insert <'Cecilia', 'F', 'Kolonsky', NULL, '1960-04-05', '6357 Windy Lane, Katy,TX', F, 28000, NULL, 4> into EMPLOYE

Result: This insertion violates the entity integrity constraint (NULL for the primary key Ssn), so it is rejected.

2) Insert <'Alicia', 'J', 'Zelaya', '999887777', '1960-04-05', '6357 Windy Lane, Katy,TX', F, 28000, '

987654321', 4> into EMPLOYEE.

Result: This insertion violates the key constraint because another tuple with the same Ssn value already exists in the EMPLOYEE relation, and so it is rejected.

3) Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windswept,Katy, TX', F, 28000,

'987654321', 7> into EMPLOYEE.

Result: This insertion violates the referential integrity constraint specified on Dno in EMPLOYEE because no corresponding referenced tuple exists inDEPARTMENT with Dnumber = 7.

4) Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windy Lane,Katy, TX', F, 28000,

NULL, 4> into EMPLOYEE.

Result: This insertion satisfies all constraints, so it is acceptable

2**) Delete operation**: This operation deletes a tuple from a relation.

- It can violate only referential integrity constraint. This occurs in case the tuple being deleted is referenced by foreign key from other tuples in the database.

Examples.

1) Delete the WORKS_ON tuple with Essn = '999887777' and Pno = 10.
   Result: This deletion is acceptable and deletes exactly one tuple.
2) Delete the EMPLOYEE tuple with Ssn = '999887777'.
   Result: This deletion is not acceptable, because there are tuples in WORKS_ON that refe to this tuple. Hence, if the tuple in EMPLOYEE is deleted, referential integrity violations will result.

3)UPDATE OPERATION: This operation is used to modify / change values of one or more attributes in tuple(s) in a relation r. It is necessary to specify a condition on attributes of a relation to select a tuple to be modified.

Updating an attribute that is neither a primary key nor a foreign key creates no problem. Modifying a primary key is similar to deleting a tuple and inserting another in its place. When updating dbms checks to confirm the new value is of correct data type and DOMAIN.

Examples:

1) Update the salary of the EMPLOYEE tuple with Ssn = '999887777' to 28000.
   Result: Acceptable.
2) Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 7.
   Result: Unacceptable, because it violates referential integrity.
3) Update the Ssn of the EMPLOYEE tuple with Ssn = '999887777' to'987654321'.
   Result: Unacceptable, because it violates primary key constraint by repeating a value that already exists as a primary key in another tuple; it violates referential integrity constraints because there are other relations that refer to the existing value of Ssn

CH-2 DATA NORMALIZATION

Informal Design Guidelines for Relation Schemas

Informal guidelines that may be used as measures to determine the quality of relation schema design:

1. Making sure that the semantics of the attributes is clear in the schema
2. Reducing the redundant information in tuples
3. Reducing the NULL values in tuples
4. Disallowing the possibility of generating spurious tuples

Imparting Clear Semantics to Attributes in Relations

The semantics of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple

Eg: Company Database

EMPLOYEE

Ename	Ssn	Bdate	Address	Dnumber

DEPARTMENT

Dname	Dnumber	Dmgr_ssn

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

Guideline 1: Design a relation schema so that it is easy to explain its meaning**.** Do not combine attributes from multiple entity types and relationship types into a single relation. Otherwise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation

For the EMP_PROJ relation in Figure 14.3(b), each tuple relates an employee to a project but also include the employee name (Ename), project name (Pname), and project location (Plocation). Although there is nothing wrong logically with these two relations, they violate Guideline 1 by mixing attributes from distinct real-world entities: EMP_DEPT mixes attributes of employees and departments, and EMP_PROJ mixes attributes of employees and projects and the WORKS_ON relationship

Redundant Information in Tuples and Update Anomalies

- One goal of schema design is to minimize the storage space used by the base relations (and hence the corresponding files).
- Grouping attributes into relation schemas has a significant effect on storage space.

For example, compare the space used by the two base relations EMPLOYEE and DEPARTMENT in above Figure with that for an EMP_DEPT base relation in above Figure .which is the result of applying the NATURAL JOIN operation to EMPLOYEE and DEPARTMENT

Storing natural joins of base relations leads to an additional problem referred to as update anomalies. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

Insertion Anomalies. Insertion anomalies can be differentiated into two types, illustrated by the following examples based on the EMP_DEPT relation:

- To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs (if the employee does not work for a department as yet).

- For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are consistent with the corresponding values for department 5 in other tuples in EMP_DEPT.
- In the design of Figure 14.2, we do not have to worry about this consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation.
- It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes for employee. This violates the entity integrity for EMP_DEPT because its primary key Ssn cannot be null.

Deletion Anomalies.

- If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost inadvertently from the database.
- This problem does not occur in the database of Figure 14.2 because DEPARTMENT tuples are stored separately.

Modification Anomalies.

- In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of all employees who work in that department; otherwise, the database will become inconsistent.
- If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong.

Guideline 2. Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly

NULL Values in Tuples

- If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples.
- Another problem with NULLs is how to account for them when aggregate operations such as COUNT or SUM are applied.
- SELECT and JOIN operations involve comparisons; if NULL values are present, the results may become unpredictable.

  NULLs can have multiple interpretations, such as the following:
- The attribute does not apply to this tuple. For example, Visa_status may not apply to U.S. students
- The attribute value for this tuple is unknown. For example, the Date_of_birth may be unknown for an employee.
- The value is known but absent; that is, it has not been recorded yet. For example, the Home_Phone_Number for an employee may exist, but may not be available and recorded yet.

Guideline 3 As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

Generation of Spurious Tuples

If we attempt a NATURAL JOIN operation on EMP_PROJ1 and EMP_LOCS, the result produces many more tuples than the original set of tuples in EMP_PROJ. In Figure 15.6, the result of applying the join to only the tuples above the dashed lines in Figure 15.5(b) is shown (to reduce the size of the resulting relation). Additional tuples that were not in EMP_PROJ are called spurious tuple

Guideline 4

Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious

tuples are generated. Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

FUNCTIONAL DEPENDENCIES AND NORMALIZATION FOR RELATIONAL DATABASES

FUNCTIONAL DEPENDENCIES: It is used to specify *formal measures* of the "goodness" of relational designs and keys are used to define **normal forms** for relations. These are **constraints** that are derived from the meaning and interrelationships of the data attributes

FD is denoted by X→Y,between two sets of attributes X and Y that are subsets of R which specifies a constraint on the possible tuples that can form a relation state R

- **The constraint is that ,For any two tuples t1 and t2 in any relation instance r(R): If $t_1[X]=t_2[X]$, *then* $t_1[Y]=t_2[Y]$**

(A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y)

- This means that the values of the Y component of a tuple in r depend on are determined by the values of the X component .
- **X -> Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y**
- X -> Y in R specifies a *constraint* on all relation instances r(R)
- FDs are derived from the real-world constraints on the attributes

Note:

1. If a constraint on R states that there cannot be more than one tuple with a given X value in any relation instance(R). i.e x is a candidate key of R,which implies that X -> Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y

2. if X->Y in R,this does not say or not Y->X in R.

Examples of FD constraints

- Social security number determines employee name. SSN -> ENAME
- Project number determines project name and location,
  PNUMBER -> {PNAME, PLOCATION}
- Employee ssn and project number determines the hours per week that the employee works on the project. {SSN, PNUMBER} -> HOURS
- An FD is a property of the attributes in the schema R
- The constraint must hold on *every* relation instance r(R)
- If K is a key of R, then K functionally determines all attributes in R
  (since we never have two distinct tuples with t1[K]=t2[K])

NORMAL FORMS BASED ON PRIMARY KEYS

NORMALIZATION OF RELATIONS

Normalization process takes a relation schema through a series of tests to certify whether it satisfies a certain form normal form.

Normalization of data is a process of analyzing the given relation schemas based on their FDs and primary key s to achieve the desirable properties

1) minimizing redundancy
2) minimizing the insertion, deletion & update anomalies

The normal form of a relation refers to the highest normal condition that it meets & hence indicates the degree to which it has been normalized

Additional properties may be needed to ensure a good relational design

i. **the lossless join or non additive join property** :which guarantees that the spurious tuple generation
ii. the **dependency preservation property**:Which ensures that each functional dependency is represented in some individual relation resulting after decomposition

Normalization: The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations

Normal form: Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

- 2NF, 3NF, BCNF :based on keys and FDs of a relation schema
- 4NF:based on keys, multi-valued dependencies : MVDs; 5NF based on keys, join dependencies : JDs

Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are *hard to understand* or to *detect*
- The database designers *need not* normalize to the highest possible normal form(usually up to 3NF, BCNF or 4NF)

Denormalization: The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

Definitions of Keys and Attributes Participating in Keys

- **A superkey of a relation schema R = {A1, A2, ..., An} is a set of attributes S *subset-of* R with the property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[S]**
- **A key K is a superkey with the *additional property* that removal of any attribute from K will cause K not to be a superkey any more.**
- The difference between a key & a superkey is that a key has to be minimal
  i.e A key k={A1,A2,…..Ak} of r,then k-{Ai} is not a key of R for any Ai,1<i<k.
- **Eg:{eno} is a key for EMPLOYEE**
  {eno},{eno,ename},{eno,ename,dob} & any set of attributes that includes eno are all superkeys.
- If a relation schema has more than one key, each is called a candidate key.

- One of the candidate keys is *arbitrarily* designated to be the primary key, and the others are called secondary keys.

- **A Prime attribute must be a member of *some* candidate key**
- **A Nonprime attribute is not a prime attribute—that is, it is not a member of any candidate key.**

FIRST NORMAL FORM:

It states that the domain of an attribute must include only atomic value & that the value of any attribute in a tuple in a tuple must be a single value from the domain of that attribute

Therefore 1 NF Disallows

- composite attributes
- multivalued attributes
- nested relations; attributes whose values for an *individual tuple* are non-atomic

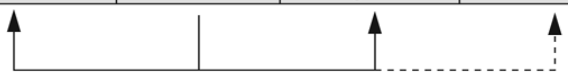
1NF : permits the attribute values which are single atomic values

Normalization into 1NF

(a)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations



(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 10.8
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

Normalization nested relations into 1NF

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, AliciaJ.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Figure 10.9

Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Example extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

SECOND NORMAL FORM :

Prime attribute: An attribute that is member of the primary key K

Full functional dependency: a FD X -> Y where removal of any attribute from X means the FD does not hold any more

i.e A belongs X,(X-{A}) does not functionally determine Y.

A FD X -> Y is a partial dependency if some attribute A belongs X, can be removed from X and the dependency still holds X,(X-{A}) ->Y

Examples:{SSN, PNUMBER} -> HOURS is a full FD since neither SSN -> HOURS nor PNUMBER -> HOURS hold

{SSN, PNUMBER} -> ENAME is not a full FD (it is called a partial dependency) since SSN -> ENAME also holds

A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key

R can be decomposed into 2NF relations via the process of 2NF normalization

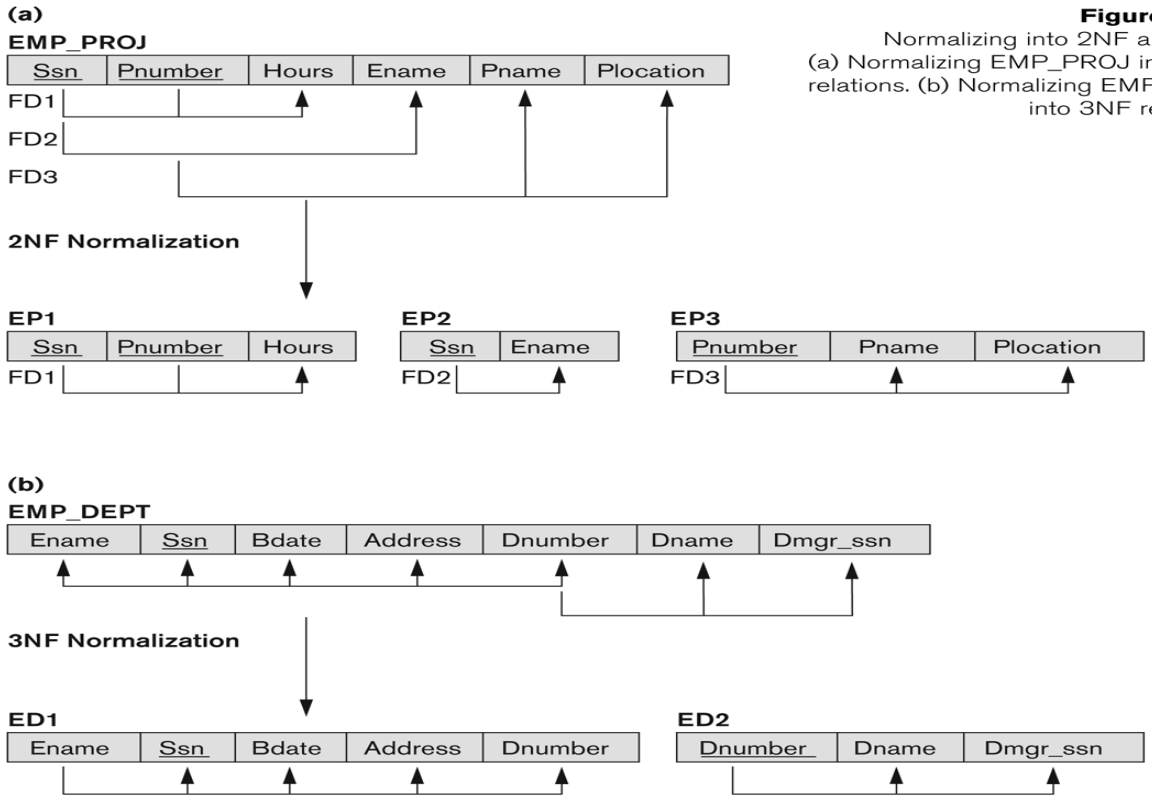


Figure 10.10
Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

THIRD NORMAL FORM

Transitive functional dependency: A FD X -> Y in a relation schema R is a Transitive functional dependency if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R, both X -> Z and Z -> Y hold

Examples:

SSN -> DMGRSSN is a **transitive** FD

Since SSN -> DNUMBER and DNUMBER -> DMGRSSN hold

SSN -> ENAME is **non-transitive**

Since there is no set of attributes X where SSN -> X and X -> ENAME

A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key

R can be decomposed into 3NF relations via the process of 3NF normalization

NOTE: In X -> Y and Y -> Z, with X as the primary key, we consider this a problem only if Y is not a candidate key.

When Y is a candidate key, there is no problem with the transitive dependency.

E.g., Consider EMP (SSN, Emp#, Salary).

Here, SSN -> Emp# -> Salary and Emp# is a candidate key.

Normal Forms Defined Informally

1st normal form:All attributes depend on the key

2nd normal form:All attributes depend on the whole key

3rd normal form:All attributes depend on nothing but the key

General Definitions Of second normal form

The above definitions consider the primary key only

A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on *every* key of R

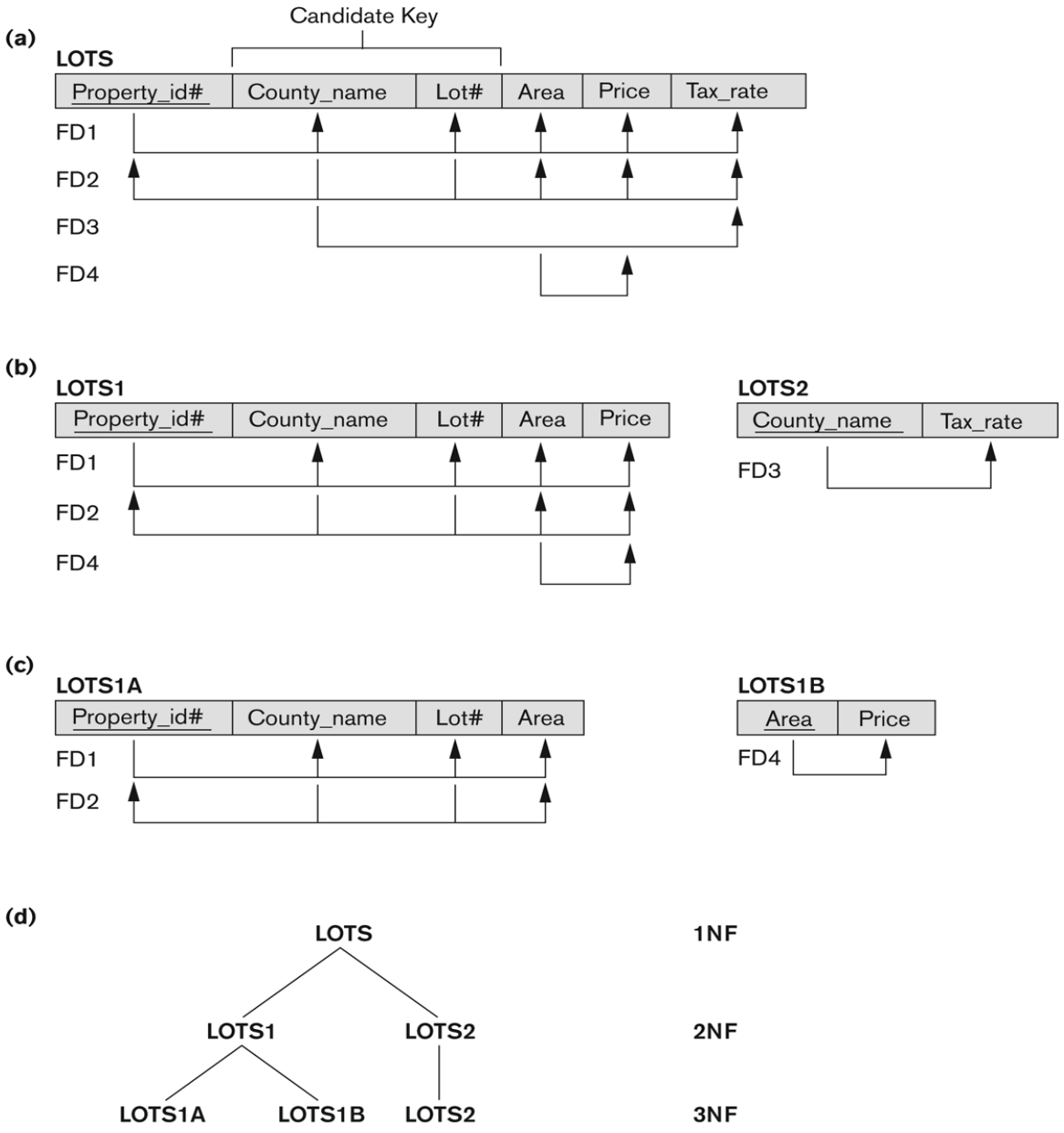


Figure 10.11
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.

General Definitions Of third normal form

Superkey of relation schema R - a set of attributes S of R that contains a key of R

A relation schema R is in third normal form (3NF) if whenever a FD X -> A holds in R, then either:

(a) X is a superkey of R, or

(b) A is a prime attribute of R

NOTE: Boyce-Codd normal form disallows condition (b) above

BCNF (Boyce-Codd Normal Form)

A relation schema R is in Boyce-Codd Normal Form (BCNF) if whenever an FD X -> A holds in R, then X is a superkey of R

Each normal form is strictly stronger than the previous one

Every 2NF relation is in 1NF

Every 3NF relation is in 2NF

Every BCNF relation is in 3NF

There exist relations that are in 3NF but not in BCNF

The goal is to have each relation in BCNF (or 3NF)

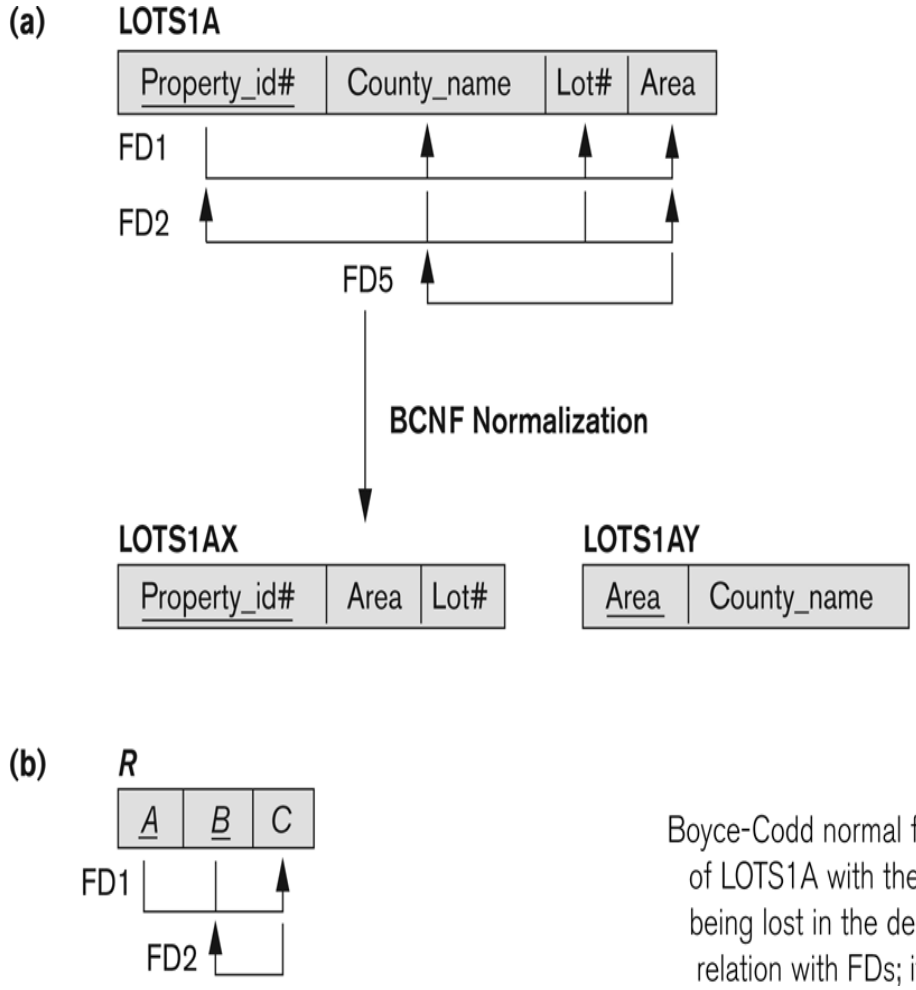


Figure 10.12
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

Figure 10.13 a relation TEACH that is in 3NF but not in BCNF

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

Figure 10.13
A relation TEACH that is in 3NF but not BCNF.

Achieving the BCNF by Decomposition

Two FDs exist in the relation TEACH:

fd1: { student, course} -> instructor

fd2: instructor -> course

{student, course} is a candidate key for this relation and that the dependencies shown follow the pattern in Figure 10.12 (b).

So this relation is in 3NF *but not in* BCNF

A relation **NOT** in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations.

Three possible decompositions for relation TEACH

1. {student, instructor} and {student, course}

2. {course, instructor } and {course, student}
3. {instructor, course } and {instructor, student}

All three decompositions will lose fd1.

- We have to settle for sacrificing the functional dependency preservation. But we cannot sacrifice the non-additivity property after decomposition.
- Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.(and hence has the non-additivity property).
- ttA test to determine whether a binary decomposition (decomposition into two relations) is non-additive (lossless)
- Verify that the third decomposition above meets the property.

Inference Rules for FDs

Given a set of FDs F, we can infer additional FDs that hold whenever the FDs in F hold

1. IR1. (Reflexive) If Y subset-of X, then X -> Y
2. IR2. (Augmentation) If X -> Y, then XZ -> YZ(Notation: XZ stands for X U Z)
3. IR3. (Transitive) If X -> Y and Y -> Z, then X -> Z

These are rules hold and all other rules that hold can be deduced from these some additional inference rules that are useful:

4. IR 4 (Decomposition): If X -> YZ, then X -> Y and X -> Z
5. IR 5 (Union): If X -> Y and X -> Z, then X -> YZ
6. IR 6 (Psuedotransitivity): If X -> Y and WY -> Z, then WX -> Z

The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

- Closure of a set F of FDs is the set F+ of all FDs that can be inferred from F
- Closure of a set of attributes X with respect to F is the set X+ of all attributes that are functionally determined by X
- X+ can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in F

EQUIVALENCE OF SETS OF FDS

Definition: A set of functional dependencies F is said to be cover another set of functional dependencies E if every FD in E is also in F^+

i.e if every dependency in E can be inferred from F ,we can say that E is covered by F

Two sets of FDs E and F are **equivalent** if:

- Every FD in E can be inferred from F, and
- Every FD in F can be inferred from E
- Hence, E and F are equivalent if both the conditions E covers F and F covers E hold

Definition(**Covers**): E **covers** F if every FD in E can be inferred from (i.e., if E^+ *subset-of* F^+)

There is an algorithm for checking equivalence of sets of FDs

Minimal Sets of Functional Dependencies

A minimal cover of a set of functional dependencies E is a set of functional dependencies F that satisfies the property that every dependency in E is in the closure F+ of F. In addition, this property is lost if any dependency from the set F is removed; F must have no redundancies in it, and the dependencies in F are in a standard form.

To satisfy these properties, we can formally define a set of functional dependencies F to be minimal if it satisfies the following conditions:

1. Every dependency in F has a single attribute for its right-hand side.

2. We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y is a proper subset of X, and still have a set of dependencies that is equivalent to F. 3. We cannot remove any dependency from F and still have a set of dependencies that is equivalent to F.

Definition. A minimal cover of a set of functional dependencies E is a minimal set of dependencies (in the standard canonical form and without redundancy) that is equivalent to E. We can always find at least one minimal cover F for any set of dependencies E.