Relational model

References

McFadden et. al. Chapter 6
Connolly and Begg Chapter 3

Rationale

The relational model was first proposed by E.F. Codd in 1970. The model's objectives were specified as follows:

- To allow a high degree of independence. Application programs must not be affected by modifications to the internal data representation, particularly by changes to file organisation, record orderings, or access paths.
- To provide substantial grounds for dealing with data semantics, constancy, and redundancy problems.
- To enable the expansion of set-oriented data manipulation languages.

Database languages

Data Definition Language (DDL)

A DDL is a language that allows the DBA or user to describe and name entities, attributes, and relationships required for the application, together with any associated integrity and security constraints.

The DDL is used to specify the database schema as asset of definitions.

Data Manipulation Language (DML)

A DML is a language that provides a set of operations to support the basic data manipulation operations on the data held in the database.

Data manipulation operations usually include the following:

- Insertion of new data into the database.
- Modification of data stored in the database.
- Retrieval of data contained in the database.
- Deletion of data contained in the database.

Basic definitions

The relational data model consists of the following three components:

- Data structure: data is organised in the form of tables with rows and columns.
- Data manipulation: powerful operations (using the SQL language) are used to manipulate data stored in a relation.

- Data integrity: facilities are included to specify business rules that maintain the integrity of data when they are manipulated.

The relational model is based on the mathematical concept of a relation, which is physically represented as a **table**. (Example of relation Fig 1)

A **relation** (also called table or file) is a table with columns and rows.
Relations are used to hold information about the objects to be held in the database.

An **attribute (**also called column or field**)** is a named column of a relation.
Attributes can appear in any order and the relation will still be the same relation, and therefore convey the same meaning.
A **domain** is the set of allowable values for one or more attributes.
Every attribute in a relation is defined on a domain. Domains may be distinct for each attribute, or two or more attributes may be defined on the same domain.
A **tuple** (also called row or record) is a row in a relation.
Tuples can appear in any order and the relation will still be the same relation.
The **degree** of a relation is the number of attributes it contains.
The **cardinality** of a relation is the number of tuples it contains.
A **relational database** is a collection of *normalised* relations with distinct relation names.

Relational data structure

The structure of a relation is expressed by a shorthand notation in which the name of the relation is followed (in brackets) by the names of the attributes. For the relation EMPLOYEE1 the notation is as follows:

```
    EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
```

It is also called the schema of the relation.

Properties (restrictions) of relations

A relation has the following properties:

- The relation has a name that is distinct from all other relation names in the relation schema.
- Each cell of the relation contains exactly one atomic (single) value. (multiple values are not allowed)
- Each attribute has a distinct name.
- The values of an attribute are all from the same domain.
- Each tuple is distinct; there are no duplicate tuples.
- The order of the attributes has no significance.
- The order of tuples has no significance, theoretically. (However, in practice, the order may affect the efficiency of accessing tuples)

Relational keys

As there are no duplicate tuples, one or more attributes are selected (called relational keys) to uniquely identify each tuple in a relation.

A **superkey** is an attribute, or set of attributes, that uniquely identifies a tuple within a relation. A superkey may contain additional attributes that are not necessary for unique identification.

A **candidate key** is a super key such that no proper subset is a superkey within the relation.
A candidate key for a relation is unique and irreducible. There may be several candidate keys for a relation.

A **primary key** is a candidate key that is selected to identify tuples uniquely within the relation. For the relation EMPLOYEE1, the primary key is underlined:

```
EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
```

A **composite key** is a primary key that consist of more than one attribute.

A **foreign key** is an attribute, or set of attributes, within one relation that matches the candidate key of some (possible the same) relation.
Consider the relations EMPLOYEE1 and DEPARTMENT,

```
EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
DEPARTMENT(Dept_Name, Location, Fax)
```

The attribute Dept_Name is a foreign key in EMPLOYEE1. It allows a user to associate any employee with the department to which they belong.

Relational integrity

Since every attribute has an associated domain, there are constraints (called domain constraints) that form restrictions on the set of values allowed for the attributes of relations.
Domain integrity means that the values that appear in a column of a relation must be taken from the same domain.

A **null** represents a value for an attribute that is currently unknown or is not applicable for this tuple.
Nulls are a way of dealing with incomplete or exceptional information. However, a null is not the same as a zero numeric value or text string filled with spaces.

In addition, there are two important integrity rules, entity integrity and referential integrity.

Entity integrity means that in a base relation, no attribute of a primary key can be null.

As the primary key of a relation is the minimal identifier that is used to identify tuples uniquely, no subset of the primary key is sufficient to provide unique identification.

Referential integrity means that if a foreign key exists in a relation, either the foreign key value must match a candidate key value of some tuple in its home relation or the foreign key value must be wholly null.

Well-structured relations

A well-structured relation contains minimal redundancy and allows users to insert, modify, and delete the rows in a table without errors and inconsistencies. Redundancies in a table may result in errors or inconsistencies (called anomalies) When a user attempts to update the data in the table, three types of anomalies are possible:

- Insertion anomaly,
- Deletion anomaly,
- Modification anomaly.

Let us consider the relation EMPLOYEE2, defined as follows:

```
EMPLOYEE2(Emp_ID, Name, Dept_Name, Salary, Course_Title,
          Date_Completed)
```

The relation has a composite key: (Emp_ID, Course_Title).
The corresponding table is in (Fig 2b)

The table holds a lot of redundant information. For example values for Emp_ID, Name, Dept_Name, and Salary appear in two separate rows for employees 100, 110 and 150. This leads to anomalies:

- Insertion anomaly: to insert a new row the user must supply values for both Emp_ID and Course_Title (since primary key values cannot be null or nonexistent). This is an anomaly since the user should be able to enter employee data without supplying course data.
- Deletion anomaly: Suppose the data from employee number 140 are deleted from the table. This will result in losing the information that this employee completed a course (Tax Acc) on 12/8/199x.
- Modification anomaly: suppose employee 100 gets a salary increase. We must record the increase in each of the rows for that employee (two occurrences), otherwise the data will be inconsistent.

Normalisation

A relation that satisfies the restrictions above is said to be in *normalised form.*

Normalisation is a technique for producing a set of relations with desirable properties, given the data requirements of an enterprise.

A normal form is the state of a relation that results from applying simple rules regarding functional dependencies (or relationships between attributes) to that relation:

- **First normal form**: any multivalued attributes (also called repeating groups) have been removed, so there is a single value (possibly null) at the intersection of each row and column.
- **Second normal form**: any partial functional dependencies have been removed.
- **Third normal form**: any transitive dependencies have been removed.
- **Boyce/Codd normal form**: any remaining anomalies that result from functional dependencies have been removed.
- **Fourth normal form**: any multivalued dependencies have been removed
- **Fifth normal form**: any remaining anomalies have been removed.

Functional dependencies and keys

Normalisation is based on the analysis of functional dependencies.
A **functional dependency** is a constraint between two attributes or sets of attributes.
For any relation R, attribute B is functionally dependent on attribute A if, for every valid instance of A, the value of A uniquely determines the value B. It is also said that B is determined by A (or A is a determinant of B).
The functional dependency of B on A is represented by an arrow, as follows: A → B.
An attribute may be functionally dependent on two (or more) attributes, rather than a single attribute.

Consider the relation
`EMP_COURSE(Emp_ID, Course_Title, Date_Completed)` (Fig 3).
The functional dependency in this relation is represented as follows:

    Emp_ID, Course_Name → Date_Completed

This implies that the date of completion of a course is completely determined by the employee and the name of the course.
Examples of functional dependency include:

1. SSN → Name, Address, Date_of_Birth. (Name, address and date of birth of a person are dependent on the social security number)
2. VIN → Make, Model, Colour. (The make, model and colour of a vehicle are functionally dependent on the vehicle identification number).
3. ISBN → Title, First_Author_Name. (the title of a book and the name of the author are functionally dependent on the ISBN of a book.

The attribute on the left-hand side of the arrow in a functional dependency is called a **determinant**. SSN, VIN and ISBN are determinants.

A candidate key must satisfy the following properties:
Unique identification. For every row, the value of the key must uniquely identify that row. ***This property implies that each non-key attribute is functionally dependent on that key.***
Non-redundancy. No attribute of that key can be deleted without destroying the property of unique identification.

In the EMPLOYEE1 relation, with schema,

```
      EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
```

Emp_ID is the only determinant on this relation, since all the other attributes are functionally dependent on it . Emp_ID is therefore a candidate key and (since there are no other candidate keys) also is the primary key.
For the relation EMPLOYEE2, Emp_Id does not uniquely identify a row in the relation. There two rows in the table for Emp_Id = 100. There are two functional dependencies in this relation:

```
      Emp_ID → Name, Dept_Name, Salary
      Emp_ID, Course_title → Date_Completed
```

The functional dependency shows that a combination of Emp_ID and Course_Title is the only candidate key (and therefore the primary key) for EMPLOYEE2. The primary key is a composite key.

A candidate key is always a determinant, while a determinant may or may not be a candidate key.

The three basic normal forms

First Normal Form

A relation is in first normal form (1NF) if it contains no multivalued attributes.
The value at the intersection of arrow and column must be atomic.

The table in Fig 2a contains multivalued attributes (repeating groups).
By removing multivalued attributes from the table it was converted into relation EMPLOYEE2.
EMPLOYEE2 is in first normal form.

Second Normal Form

A relation is in second normal form (2NF) if it is in first normal form and every non-key attribute is fully functionally dependent on the primary key.
A relation that is in first normal form will be in second normal form if any one of the following conditions applies:

1. The primary key consists of only one attribute (such as the attribute Emp_ID in EMPLOYEE1)
2. No non-key attributes exist in the relation (thus all of the attributes in the relation are components of the primary key)
3. Every non-key attribute is functionally dependent on the full set of primary key attributes.

EMPLOYEE2 (Fig 2b) is an example of relation that is not in second normal form. The primary key for this relation is the composite key (Emp_ID, Course_Title). Non-key attributes Name, Dept_name and Salary are functionally dependent on part of the primary key (Emp_ID) but not on Course_Title.

A partial functional dependency in EMPLOYEE2 is a functional dependency in which one or more non-key attributes (such as Name) are functionally dependent on part (but not all) of the primary key.

To convert a relation to second normal form, we decompose the relation into new relations that satisfy one (or more) of the conditions described above. EMPLOYEE2 is decomposed into the following two relations:
EMPLOYWEE1 (Emp_ID, Name, Dept,_Name, Salary). This relation is both in 1NF and 2NF. (Fig 1)
EMP_COURSE(Emp_ID, Course_Title, Date_Completed). This relation satisfies is in 2NF (Fig 3).

Third Normal Form

A relation is in **third normal form** (3NF) if it is in 2NF and no transitive dependencies exist.
A **transitive dependency** in a relation is a functional dependency between two (or more) non-key attributes. If B is functionally dependent on A, and C is functionally dependent on B, then C is transitively dependent on A.
For example, consider the relation
SALES(Cust_ID, Name, Salesperson, Region) (see Fig 4)

Cust_ID is the primary key, so that the remaining attributes are functionally dependent on this attribute. There is, however, a transitive dependency: Region is functionally dependent on Salesperson and Salesperson is functionally dependent on Cust_ID. As a result there are update anomalies:
Insertion anomaly: a salesperson assigned to a region cannot be entered until a customer has been assigned to that salesperson.
Deletion anomaly: if a customer is deleted from the table, we lose the information that a salesperson is assigned to a region.
Update anomaly: if a salesperson is reassigned to a new region several rows must be changed to reflect that fact.

A normalised version is given in Fig 5.

Subtypes (see Conceptual)

Relational model

References

McFadden et. al. Chapter 6
Connolly and Begg Chapter 3

Rationale

The relational model was first proposed by E.F. Codd in 1970. The model's objectives were specified as follows:

- To allow a high degree of independence. Application programs must not be affected by modifications to the internal data representation, particularly by changes to file organisation, record orderings, or access paths.
- To provide substantial grounds for dealing with data semantics, constancy, and redundancy problems.
- To enable the expansion of set-oriented data manipulation languages.

Database languages

Data Definition Language (DDL)

A DDL is a language that allows the DBA or user to describe and name entities, attributes, and relationships required for the application, together with any associated integrity and security constraints.

The DDL is used to specify the database schema as asset of definitions.

Data Manipulation Language (DML)

A DML is a language that provides a set of operations to support the basic data manipulation operations on the data held in the database.

Data manipulation operations usually include the following:

- Insertion of new data into the database.
- Modification of data stored in the database.
- Retrieval of data contained in the database.
- Deletion of data contained in the database.

Basic definitions

The relational data model consists of the following three components:

- Data structure: data is organised in the form of tables with rows and columns.
- Data manipulation: powerful operations (using the SQL language) are used to manipulate data stored in a relation.
- Data integrity: facilities are included to specify business rules that maintain the integrity of data when they are manipulated.

The relational model is based on the mathematical concept of a relation, which is physically represented as a **table**. (Example of relation Fig 1)

A **relation** (also called table or file) is a table with columns and rows.

Relations are used to hold information about the objects to be held in the database.

An **attribute** (also called column or field) is a named column of a relation.
Attributes can appear in any order and the relation will still be the same relation, and therefore convey the same meaning.
A **domain** is the set of allowable values for one or more attributes.
Every attribute in a relation is defined on a domain. Domains may be distinct for each attribute, or two or more attributes may be defined on the same domain.
A **tuple** (also called row or record) is a row in a relation.
Tuples can appear in any order and the relation will still be the same relation.
The **degree** of a relation is the number of attributes it contains.
The **cardinality** of a relation is the number of tuples it contains.
A **relational database** is a collection of *normalised* relations with distinct relation names.

Relational data structure

The structure of a relation is expressed by a shorthand notation in which the name of the relation is followed (in brackets) by the names of the attributes. For the relation EMPLOYEE1 the notation is as follows:

```
EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
```

It is also called the schema of the relation.

Properties (restrictions) of relations

A relation has the following properties:

- The relation has a name that is distinct from all other relation names in the relation schema.
- Each cell of the relation contains exactly one atomic (single) value. (multiple values are not allowed)
- Each attribute has a distinct name.
- The values of an attribute are all from the same domain.
- Each tuple is distinct; there are no duplicate tuples.
- The order of the attributes has no significance.
- The order of tuples has no significance, theoretically. (However, in practice, the order may affect the efficiency of accessing tuples)

Relational keys

As there are no duplicate tuples, one or more attributes are selected (called relational keys) to uniquely identify each tuple in a relation.

A **superkey** is an attribute, or set of attributes, that uniquely identifies a tuple within a relation. A superkey may contain additional attributes that are not necessary for unique identification.

A **candidate key** is a super key such that no proper subset is a superkey within the relation.
A candidate key for a relation is unique and irreducible. There may be several candidate keys for a relation.

A **primary key** is a candidate key that is selected to identify tuples uniquely within the relation. For the relation EMPLOYEE1, the primary key is underlined:

```
EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
```

A **composite key** is a primary key that consist of more than one attribute.

A **foreign key** is an attribute, or set of attributes, within one relation that matches the candidate key of some (possible the same) relation.
Consider the relations EMPLOYEE1 and DEPARTMENT,

```
EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
DEPARTMENT(Dept_Name, Location, Fax)
```

The attribute Dept_Name is a foreign key in EMPLOYEE1. It allows a user to associate any employee with the department to which they belong.

Relational integrity

Since every attribute has an associated domain, there are constraints (called domain constraints) that form restrictions on the set of values allowed for the attributes of relations.
Domain integrity means that the values that appear in a column of a relation must be taken from the same domain.

A **null** represents a value for an attribute that is currently unknown or is not applicable for this tuple.
Nulls are a way of dealing with incomplete or exceptional information. However, a null is not the same as a zero numeric value or text string filled with spaces.

In addition, there are two important integrity rules, entity integrity and referential integrity.

Entity integrity means that in a base relation, no attribute of a primary key can be null.

As the primary key of a relation is the minimal identifier that is used to identify tuples uniquely, no subset of the primary key is sufficient to provide unique identification.

Referential integrity means that if a foreign key exists in a relation, either the foreign key value must match a candidate key value of some tuple in its home relation or the foreign key value must be wholly null.

Well-structured relations

A well-structured relation contains minimal redundancy and allows users to insert, modify, and delete the rows in a table without errors and inconsistencies. Redundancies in a table may result in errors or inconsistencies (called anomalies) When a user attempts to update the data in the table. Three types of anomalies are possible:

- Insertion anomaly,
- Deletion anomaly,
- Modification anomaly.

Let us consider the relation EMPLOYEE2, defined as follows:

```
EMPLOYEE2(Emp_ID, Name, Dept_Name, Salary, Course_Title,
          Date_Completed)
```

The relation has a composite key: (Emp_ID, Course_Title).
The corresponding table is in (Fig 2b)

The table holds a lot of redundant information. For example values for Emp_ID, Name, Dept_Name, and Salary appear in two separate rows for employees 100, 110 and 150. This leads to anomalies:

- Insertion anomaly: to insert a new row the user must supply values for both Emp_ID and Course_Title (since primary key values cannot be null or nonexistent). This is an anomaly since the user should be able to enter employee data without supplying course data.
- Deletion anomaly: Suppose the data from employee number 140 are deleted from the table. This will result in losing the information that this employee completed a course (Tax Acc) on 12/8/199x.
- Modification anomaly: suppose employee 100 gets a salary increase. We must record the increase in each of the rows for that employee (two occurrences), otherwise the data will be inconsistent.

Normalisation

A relation that satisfies the restrictions above is said to be in normalised form.

Normalisation is a technique for producing a set of relations with desirable properties, given the data requirements of an enterprise.

A normal form is state of a relation that results from applying simple rules regarding functional dependencies (or relationships between attributes) to that relation:

- **First normal form**: any multivalued attributes (also called repeating groups) have been removed, so there is a single value (possibly null) at the intersection of each row and column.
- **Second normal form**: any partial functional dependencies have been removed.
- **Third normal form**: any transitive dependencies have been removed.
- **Boyce/Codd normal form**: any remaining anomalies that result from functional dependencies have been removed.

- **Fourth normal form**: any multivalued dependencies have been removed
- **Fifth normal form**: any remaining anomalies have been removed.

Functional dependencies and keys

Normalisation is based on the analysis of functional dependencies.
A **functional dependency** is a constraint between two attributes or sets of attributes.
For any relation R, attribute B is functionally dependent on attribute A if, for every valid instance of A, the value of A uniquely determines the value B. It is also said that B is determined by A (or A is a determinant of B).
The functional dependency of B on A is represented by an arrow, as follows: A → B.
An attribute may be functionally dependent on two (or more) attributes, rather than a single attribute.

Consider the relation
`EMP_COURSE(Emp_ID, Course_Title, Date_Completed)` (Fig 3).
The functional dependency in this relation is represented as follows:

```
      Emp_ID, Course_Name →  Date_Completed
```

This implies that the date of completion of a course is completely determined by the employee and the name of the course.
Examples of functional dependency include:

4. SSN → Name, Address, Date_of_Birth. (Name, address and date of birth of a person are dependent on the social security number)
5. VIN → Make, Model, Colour. (The make, model and colour of a vehicle are functionally dependent on the vehicle identification number).
6. ISBN → Title, First_Author_Name. (the title of a book and the name of the author are functionally dependent on the ISBN of a book.

The attribute on the left-hand side of the arrow in a functional dependency is called a **determinant**. SSN, VIN and ISBN are determinants.

A candidate key must satisfy the following properties:
Unique identification. For every row, the value of the key must uniquely identify that row. ***This property implies that each nonkey attribute is functionally dependent on that key.***
Non-redundancy. No attribute of that key can be deleted without destroying the property of unique identification.

In the EMPLOYEE1 relation, with schema,

```
      EMPLOYEE1(Emp_ID, Name, Dept_Name, Salary)
```

Emp_ID is the only determinant on this relation, since all the other attributes are functionally dependent on it . Emp_ID is therefore a candidate key and (since there are no other candidate keys) also is the primary key.
For the relation EMPLOYEE2, Emp_Id does not uniquely identify a row in the relation. There two rows in the table for Emp_Id 100. There are two functional dependencies in this relation:

```
Emp_ID → Name, Dept_Name, Salary
Emp_ID, Course_title → Date_Completed
```

The functional dependency shows that a combination of Emp_ID and Course_Title is the only candidate key (and therefore the primary key) for EMPLOYEE2. The primary key is a composite key.

A candidate key is always a determinant, while a determinant may or may not be a candidate key.

The three basic normal forms

First Normal Form

A relation is in first normal form (1NF) if it contains no multivalued attributes.
The value at the intersection of arrow and column must be atomic.

The table in Fig 2a contains multivalued attributes (repeating groups).
By removing multivalued attributes from the table it was converted into relation EMPLOYEE2.
EMPLOYEE2 is in first normal form.

Second Normal Form

A relation is in second normal form (2NF) if it is in first normal form and every nonkey attribute is fully functionally dependent on the primary key.
A relation that is in first normal form will be in second normal form if any one of the following conditions applies:

4. The primary key consists of only one attribute (such as the attribute Emp_ID in EMPLOYEE1)
5. Non nonkey attributes exist in the relation (thus all of the attributes in the relation are components of the primary key)
6. Every nonkey attribute is functionally dependent on the full set of primary key attributes.

EMPLOYEE2 (Fig 2b) is an example of relation that is not in second normal form. The primary key for this relation is the composite key Emp_ID, Course_Title. Nonkey attributes Name, Dept_name and Salary are functionally dependent on part of the primary key (Emp_ID) but not on Course_Title.
A partial funtional dependency in EMPLOYEE2 is a functional dependency in which one or more nonkey attributes (such as Name) are functionally dependent on part (but not all) of the primary key.

To convert a relation to second normal form, we decompose the relation into new relations that satisfy one (or more) of the conditions described above. EMPLOYEE2 is decomposed into the following two relations:
EMPLOYWEE1 (<u>Emp_ID</u>, Name, Dept,_Name, Salary). This relation is both in 1NF and 2NF. (Fig 1)

EMP_COURSE(Emp_ID, Course_Title, Date_Completed). This relation satisfies is in 2NF (Fig 3).

Third Normal Form

A relation is in **third normal form** (3NF) if it is in 2NF and no transitive dependencies exist.
A **transitive dependency** in a relation is a functional dependency between two (or more) nonkey attributes. If B is functionally dependent on A, and C is functionally dependent on B, then C is transitively dependent on A.
For example, consider the relation

SALES(Cust_ID, Name, Salesperson, Region) (see Fig 4)

Cust_ID is the primary key, so that the remaining attributes are functionally dependent on this attribute. There is, however, a transitive dependency: Region is functionally dependent on Salesperson and Salesperson is functionally dependent on Cust_ID. As a result there are update anomalies:
Insertion anomaly: a salesperson assigned to a region cannot be entered until a customer has been assigned to that salesperson.
Deletion anomaly: if a customer is deleted from the table, we lose the information that a salesperson is assigned to a region.
Update anomaly: if a salesperson is reassigned to a new region several rows must be changed to reflect that fact.

A normalised version is given in Fig 5.

EMPLOYEE1

Emp_ID	Name	Dept_Name	Salary
100	Margaret Simpson	Marketing	48,000
140	Allen Beeton	Accounting	52,000
110	Chris Lucero	Info Systems	43,000
190	Lorenzo Davis	Finance	55,000
150	Susan Martin	Marketing	42,000

Figure 1
EMPLOYEE1 relation with sample data

Figure 2 - [illegible]
Eliminating multivalued attributes

(a) Table with repeating groups

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/199X
				Surveys	10/7/199X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/199X
110	Chris Lucero	Info Systems	43,000	SPSS	1/12/199X
				C++	4/22/199X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/16/199X
				Java	8/12/199X

(b) EMPLOYEE2 relation

EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/199X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/199X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/199X
110	Chris Lucero	Info Systems	43,000	SPSS	1/12/199X
110	Chris Lucero	Info Systems	43,000	C++	4/22/199X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/199X
150	Susan Martin	Marketing	42,000	Java	8/12/199X

Emp_ID	Course_Title	Date_Completed
100	SPSS	6/19/199X
100	Surveys	10/7/199X
140	Tax Acc	12/8/199X
110	SPSS	1/12/199X
110	C++	4/22/199X
150	SPSS	6/19/199X
150	Java	8/12/199X

Figure 3
EMP_COURSE

Figure 4
Relation with transitive dependency

(a) SALES relation with sample data

SALES

Cust_ID	Name	Salesperson	Region
8023	Anderson	Smith	South
9167	Bancroft	Hicks	West
7924	Hobbs	Smith	South
6837	Tucker	Hernandez	East
8596	Eckersley	Hicks	West
7018	Arnold	Faulb	North

(b) Transitive dependency in SALES relation

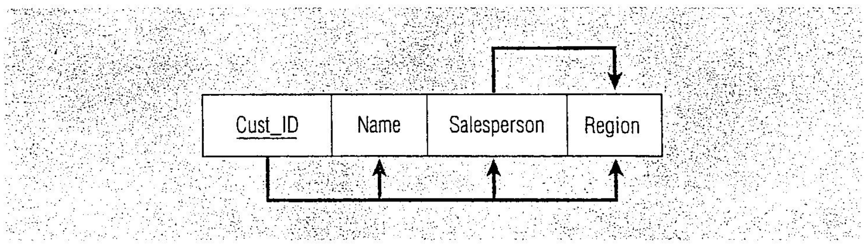


Figure 5
Removing a transitive dependency

(a) Decomposing the SALES relation

SALES1

Cust_ID	Name	Salesperson
8023	Anderson	Smith
9167	Bancroft	Hicks
7924	Hobbs	Smith
6837	Tucker	Hernandez
8596	Eckersley	Hicks
7018	Arnold	Faulb

SPERSON

Salesperson	Region
Smith	South
Hicks	West
Hernandez	East
Faulb	North

(b) Relations in 3NF

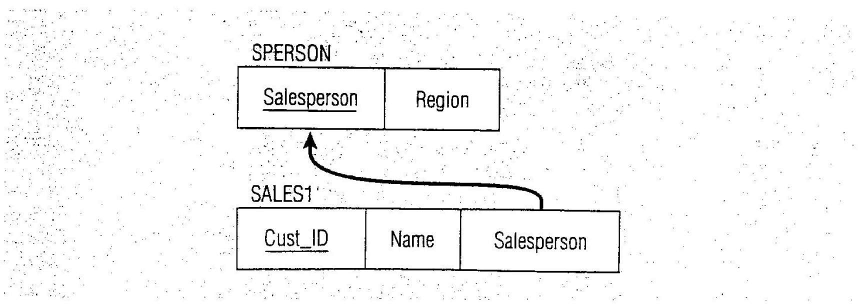


Fig. 6.14: (initial, badly-normalised table):
Table-b-n ($\underline{\text{A, B,}}$ C, D, E, F, G)

Fig. 6.14: (final, well-normalised tables):
Table-w-n-1 ($\underline{\text{A, B}}$, F)
Table-w-n-2 ($\underline{\text{A}}$, C)
Table-w-n-3 ($\underline{\text{C}}$, D, E)
Table-w-n-4 ($\underline{\text{F}}$, G)

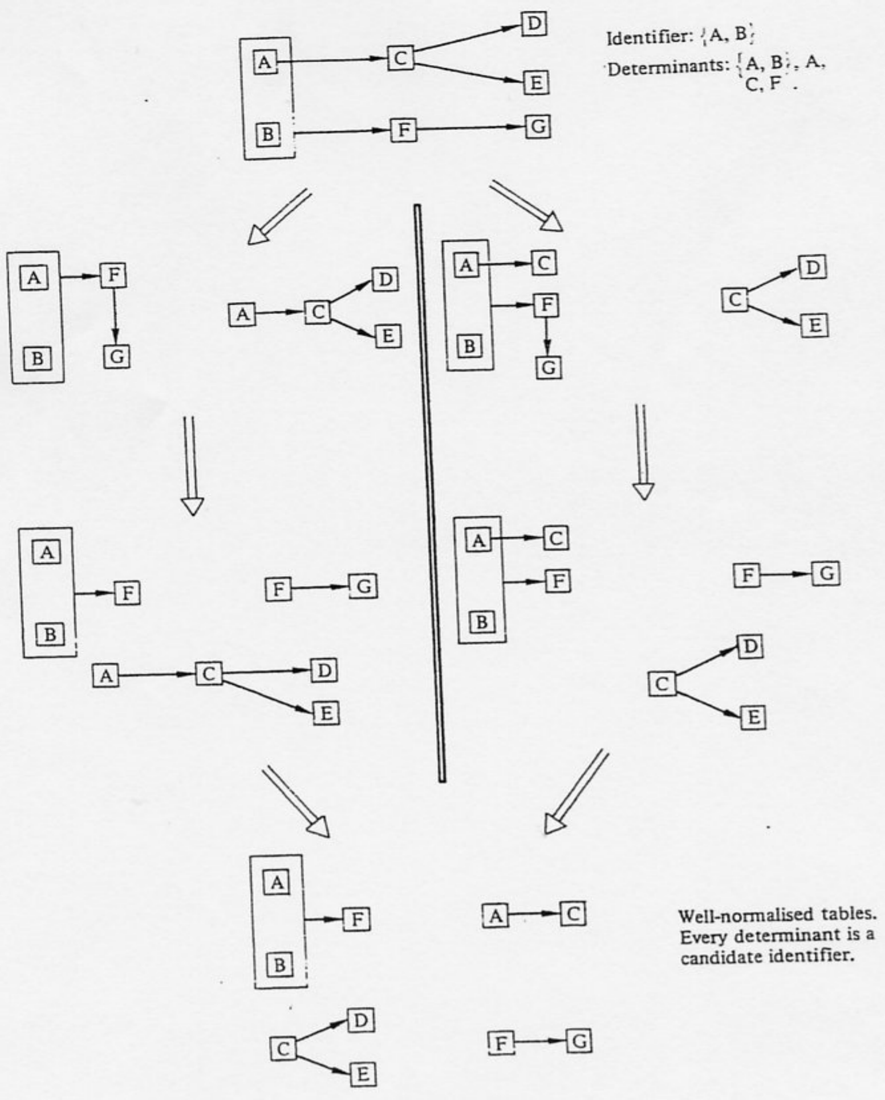

Identifier: {A, B}
Determinants: {A, B}, A, C, F.
Well-normalised tables. Every determinant is a candidate identifier.

Fig. 6.14 Two routes to the same well-normalised solution

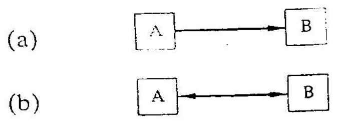
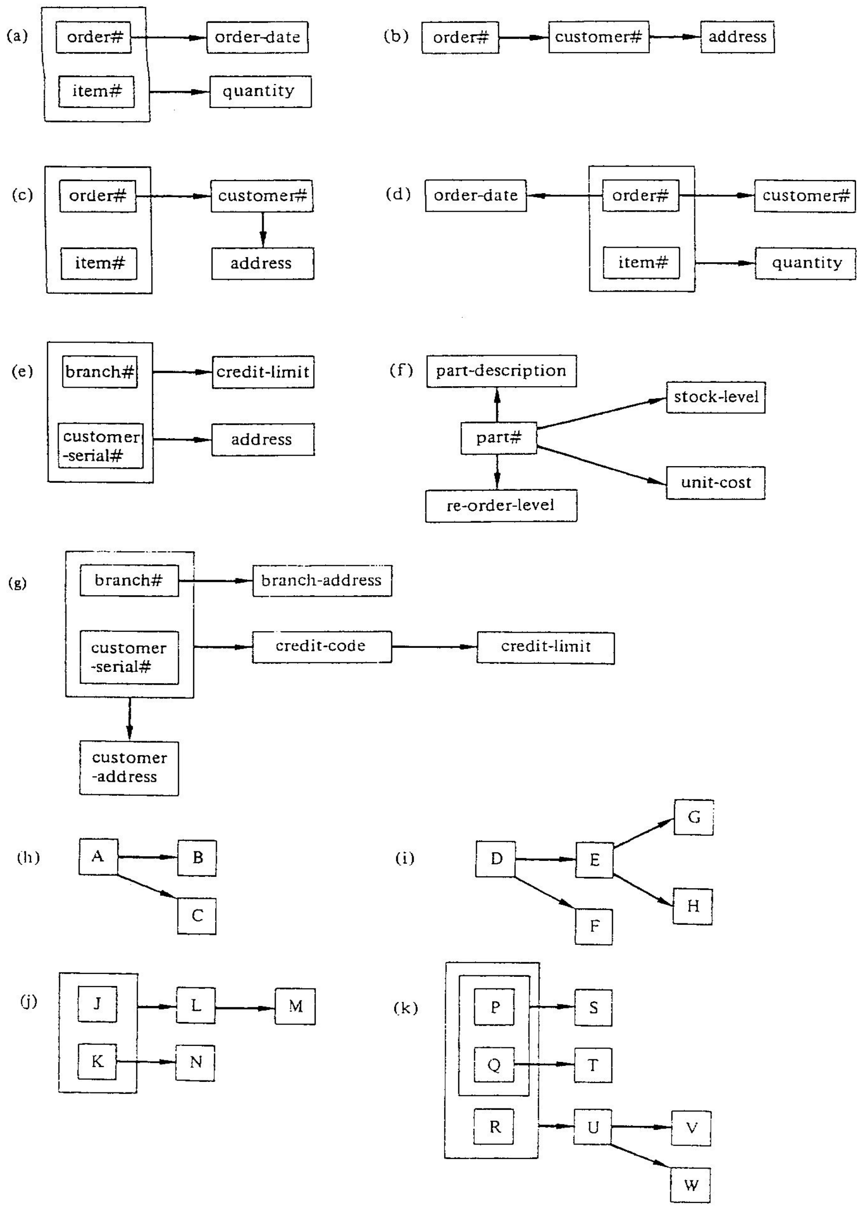


Fig. Determinancy diagrams
(a) Attribute A is a determinant of attribute B
(b) Attribute A is a determinant of attribute B, and attribute B is a determinant of attribute A

Questions

1. For each of the following determinancy diagrams, identify each determinant and the attribute(s) it directly determines.


(a)
order#
order-date
item#
quantity
(b)
order#
customer#
address
(c)
order#
customer#
item#
address
(d)
order-date
order#
customer#
item#
quantity
(e)
branch#
credit-limit
customer -serial#
address
(f)
part-description
stock-level
part#
unit-cost
re-order-level
(g)
branch#
branch-address
customer -serial#
credit-code
credit-limit
customer -address
(h)
A
B
C
(i)
D
E
G
H
F
(j)
J
L
M
K
N
(k)
P
S
Q
T
R
U
V
W