Why are entity integrity and referential integrity constraints considered important?

Use of Entity Integrity Constraint:

1. **Uniqueness:** Entity integrity ensures that each record can be uniquely identified by a primary key. This prevents duplicate records and maintains the distinctiveness of each entity within a table.
2. **Non-nullability:** Primary keys cannot be null, ensuring that every record has a valid identifier. This is crucial for accurately linking records across different tables.
3. **Data Integrity:** By guaranteeing that each entity has a unique and non-null primary key, entity integrity helps maintain accurate and reliable data within the database.

Use of Referential Integrity Constraint:

1. **Consistency:** Referential integrity ensures that a foreign key in one table correctly references a primary key in another table. This maintains the consistency between tables.
2. **Data Accuracy:** By ensuring that relationships between tables are valid and consistent, referential integrity helps maintain the accuracy and reliability of data throughout the database.

Explain the theory of functional dependency. Why functional dependency is important in relational database design?

Functional dependency specifies the relationship between two sets of attributes where one attribute determines the value of another attribute. It is denoted as **X → Y**, where the attribute set on the left side of the arrow, **X** is called **Determinant**, and **Y** is called the **Dependent**.

Uses of functional dependency:

1. Data Normalization: With the help of functional dependencies we are able to identify the primary key, candidate key in a table which in turns helps in normalization.

2. Query Optimization: With the help of functional dependencies we are able to decide the connectivity between the tables and perform query optimization to improve performance.

3. Consistency of Data: Functional dependencies ensures the consistency of the data by removing any redundancies or inconsistencies that may exist in the data.

4. Data Quality Improvement: Functional dependencies ensure that the data in the database to be accurate, complete and updated.

Explain the various relational algebra operations with suitable examples.

1. **Selection(σ):** It is used to select required tuples of the relations.

Example:

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

R

A	B	C
1	2	4
4	3	4

σ(C>3)R

2. **Projection(π):** It is used to project required column data from a relation.

Example:

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

R

B	C
2	4
2	3
3	4

π(B,C)R

3. **Union:** Union of sets A and B, denoted by **A ∪ B,** is the set of distinct elements that belong to set A or set B, or both.

   Example:

Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

FRENCH

Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21
Rohan	25

GERMAN

Student_Name
Ram
Mohan
Vivek
Geeta
Shyam
Rohan

```
π(Student_Name)FRENCH U π(Student_Name)GERMAN
```

4. **Set Difference:** It is denoted by **'A – B'**, means the set containing elements that are in A but not in B.
   Example: In the previous table:

Student_Name
Ram
Mohan

```
π(Student_Name)FRENCH - π(Student_Name)GERMAN
```

5. **Set Intersection:** The intersection of the sets A and B, denoted by **A ∩ B,** is the set of elements that belong to both A and B.
   Example:

Student_Name
Vivek
Geeta

$\pi(Student\_Name)FRENCH \cap \pi(Student\_Name)GERMAN$

6. **Rename:** Rename is a unary operation used for renaming attributes of a relation.

Example:

ρ(a/b)R will rename the attribute 'b' of the relation by 'a'.

7. **Cross Product:** Cross product A X B will result in all the attributes of A followed by each attribute of B.
   Example:

Name	Age	Sex
Ram	14	M
Sona	15	F
Kim	20	M

A

ID	Course
1	DS
2	DBMS

B

Name	Age	Sex	ID	Course
Ram	14	M	1	DS
Ram	14	M	2	DBMS
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Kim	20	M	1	DS
Kim	20	M	2	DBMS

A X B

Discuss the options for mapping EER model constructs to relations.

Step8: Options for Mapping Specialization or Generalization.

- Convert each specialization with m subclasses {S1, S2,….,Sm} and generalized superclass C, where the attributes of C are {k,a1,…an} and k is the (primary) key, into relational schemas using one of the four following options:
  - Option 8A: Multiple relations-Superclass and subclasses
  - Option 8B: Multiple relations-Subclass relations only
  - Option 8C: Single relation with one type attribute
  - Option 8D: Single relation with multiple type attributes

- **Option 8A: Multiple relations-Superclass and subclasses**
  - Create a relation L for C with attributes Attrs(L) = {k,a1,…an} and PK(L) = k. Create a relation Li for each subclass Si, 1 < i < m, with the attributesAttrs(Li) = {k} U {attributes of Si} and PK(Li)=k. This option works for any specialization (total or partial, disjoint of over-lapping).
- **Option 8B: Multiple relations-Subclass relations only**
  - Create a relation Li for each subclass Si, 1 < i < m, with the attributes Attr(Li) = {attributes of Si} U {k,a1…,an} and PK(Li) = k. This option only works for a specialization whose subclasses are total (every entity in the superclass must belong to (at least) one of the subclasses).

- **Option 8C: Single relation with one type attribute**
  - Create a single relation L with attributes Attrs(L) = {k,a_1,…a_n} U {attributes of S_1} U…U {attributes of S_m} U {t} and PK(L) = k. The attribute t is called a type (or **discriminating**) attribute that indicates the subclass to which each tuple belongs
- **Option 8D: Single relation with multiple type attributes**
  - Create a single relation schema L with attributes Attrs(L) = {k,a_1, …a_n} U {attributes of S_1} U…U {attributes of S_m} U {t_1, t_2,…,t_m} and PK(L) = k. Each t_i, 1 < I < m, is a Boolean type attribute indicating whether a tuple belongs to the subclass S_i.

Step 9: Mapping of Union Types (Categories).

- For mapping a category whose defining superclass have different keys, it is customary to specify a new key attribute, called a surrogate key, when creating a relation to correspond to the category.

What is meant by the term heuristic optimization? Discuss the main heuristics that are applied during query optimization with an example.

Process for heuristics optimization

- The parser of a high-level query generates an initial internal representation;
- Apply heuristics rules to optimize the internal representation.
- A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.

The main heuristic is to:

1. Apply the operations that reduce the size of intermediate results.
2. Perform select operations as early as possible to reduce the number of tuples and perform project operations as early as possible to reduce the number of attributes. (This is done by moving select and project operations as far down the tree as possible.)
3. The select and join operations that are most restrictive should be executed before other similar operations. (This is done by reordering the leaf nodes of the tree among themselves and adjusting the rest of the tree appropriately.)

Example:

```
Q: SELECT   LNAME
   FROM     EMPLOYEE, WORKS_ON, PROJECT
   WHERE    PNAME = 'AQUARIUS' AND
            PNMUBER=PNO AND ESSN=SSN
            AND BDATE > '1957-12-31';
```

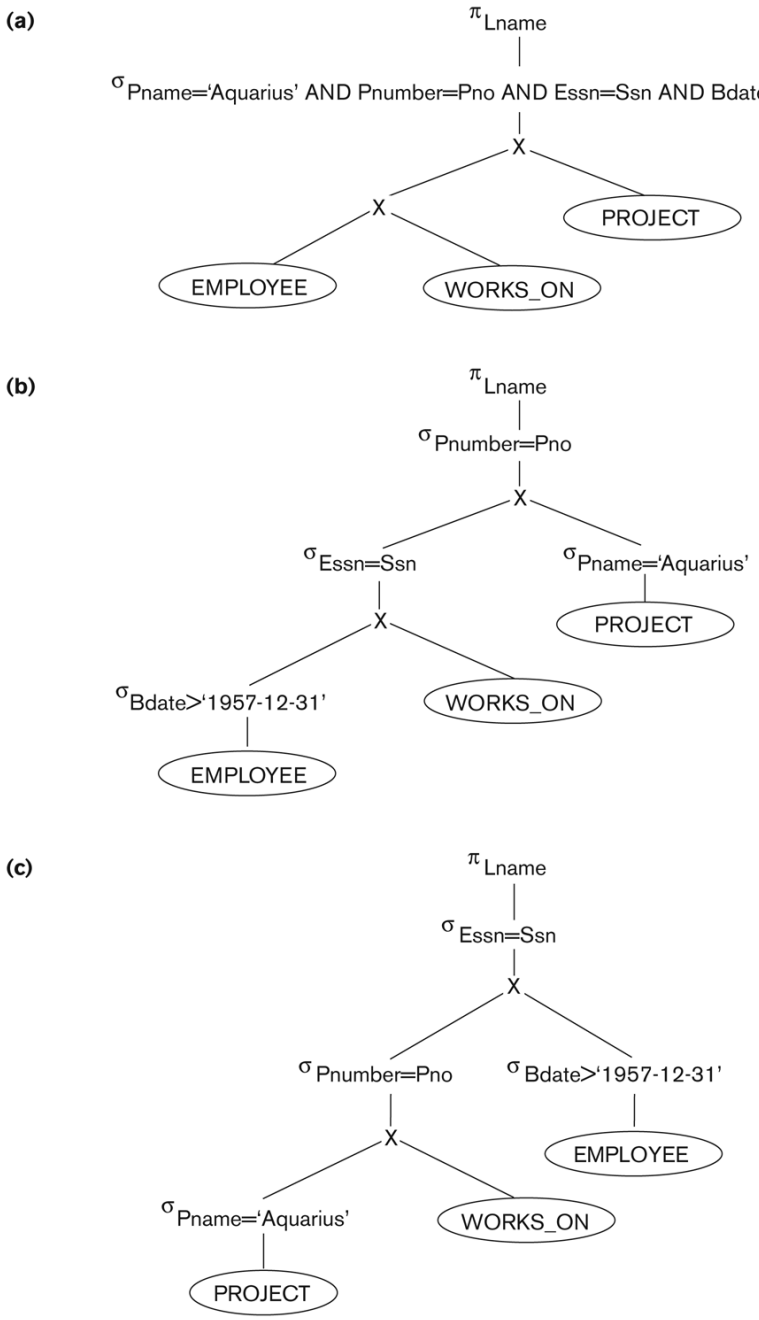
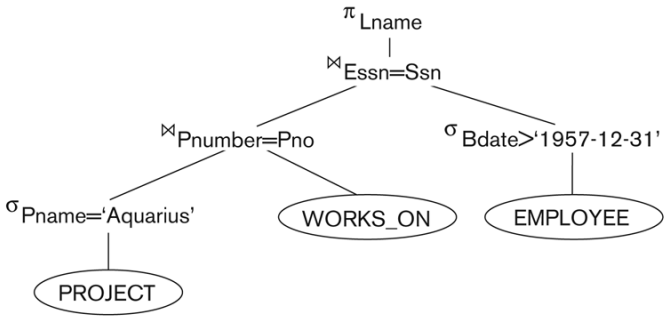


Figure 15.5
Steps in converting a query tree during heuristic optimization.
(a) Initial (canonical) query tree for SQL query Q.
(b) Moving SELECT operations down the query tree.
(c) Applying the more restrictive SELECT operation first.
(d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.
(e) Moving PROJECT operations down the query tree.

(d)



(e)

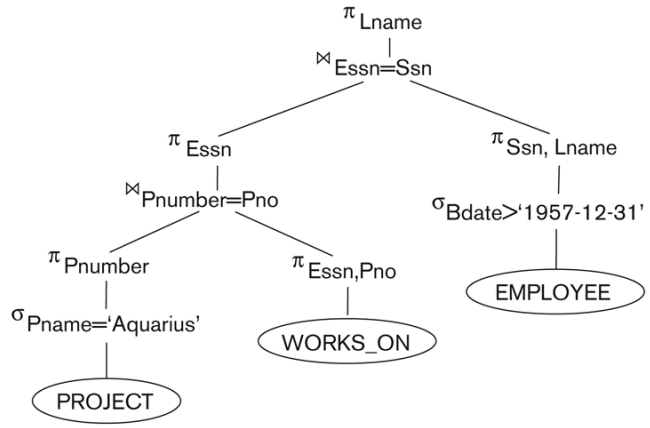


Figure 15.5
Steps in converting a query tree during heuristic optimization.
(a) Initial (canonical) query tree for SQL query Q.
(b) Moving SELECT operations down the query tree.
(c) Applying the more restrictive SELECT operation first.
(d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.
(e) Moving PROJECT operations down the query tree.

How does a query tree represent a relational algebra expression? What is meant by execution of a query tree? Discuss the rules for transformation of query tree.

Query tree: A tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as **leaf nodes** of the **tree**, and represents the relational algebra operations as internal nodes.

An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation.

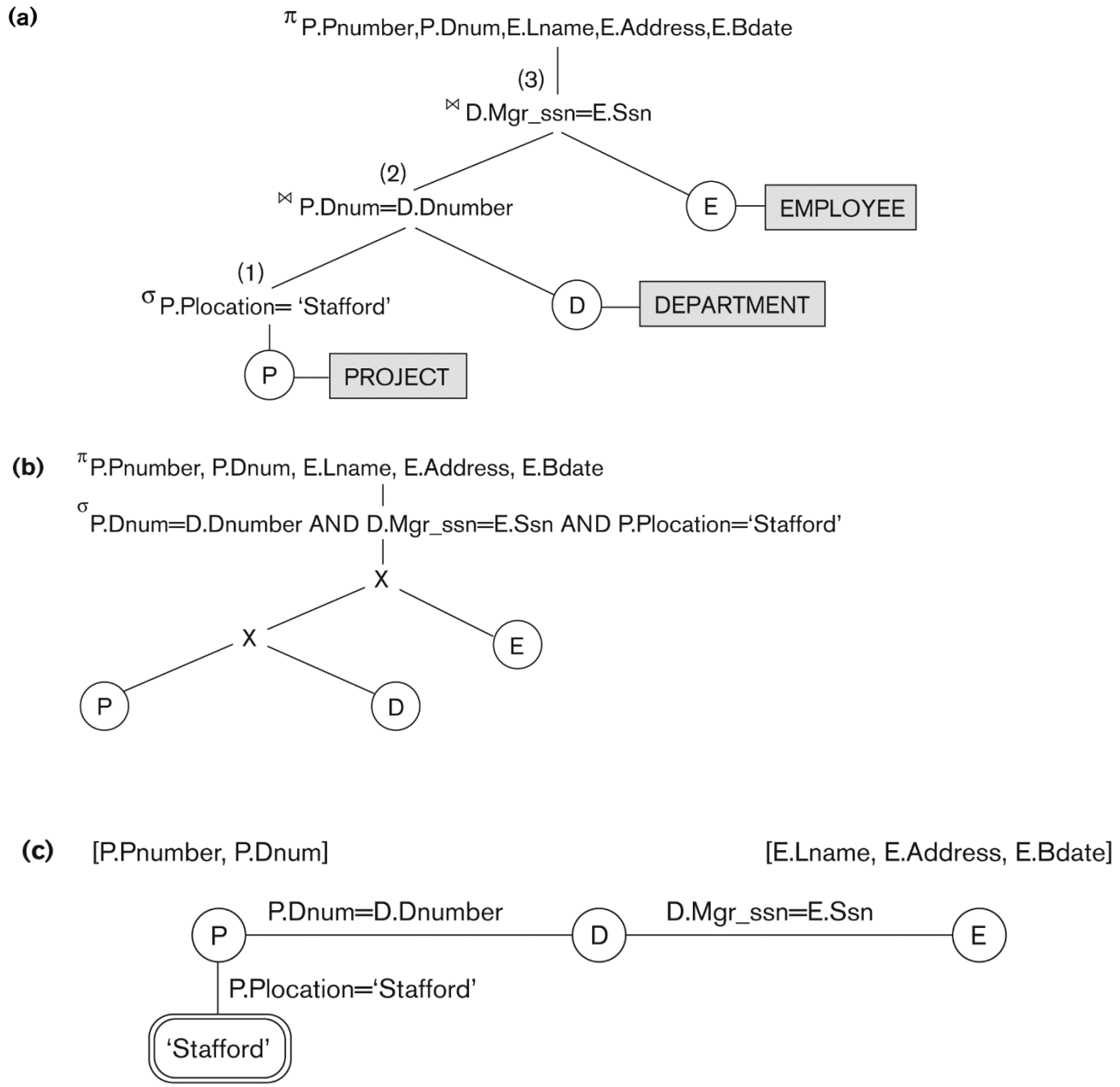


Figure 15.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

- General Transformation Rules for Relational Algebra Operations:

1. Cascade of σ: A conjunctive selection condition can be broken up into a cascade (sequence) of individual σ operations:
   - $\sigma_{c1 \text{ AND } c2 \text{ AND } \ldots \text{ AND } cn}(R) = \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))\ldots))$
2. Commutativity of σ: The σ operation is commutative:
   - $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$
3. Cascade of π: In a cascade (sequence) of π operations, all but the last one can be ignored:
   - $\pi_{List1}(\pi_{List2}(\ldots(\pi_{Listn}(R))\ldots)) = \pi_{List1}(R)$
4. Commuting σ with π: If the selection condition c involves only the attributes A1, ..., An in the projection list, the two operations can be commuted:
   - $\pi_{A1, A2, \ldots, An}(\sigma_c(R)) = \sigma_c(\pi_{A1, A2, \ldots, An}(R))$
5. Commutativity of ⋈(and x): The ⋈ operation is commutative as is the x operation:
   - $R \bowtie_C S = S \bowtie_C R$; $R \times S = S \times R$
6. Commuting σ with ⋈ (or x): If all the attributes in the selection condition c involve only the attributes of one of the relations being joined—say, R—the two operations can be commuted as follows:
   - $\sigma_c(R \bowtie S) = (\sigma_c(R)) \bowtie S$
   - Alternatively, if the selection condition c can be written as (c1 and c2), where condition c1 involves only the attributes of R and condition c2 involves only the attributes of S, the operations commute as follows:
   - $\sigma_c(R \bowtie S) = (\sigma_{c1}(R)) \bowtie (\sigma_{c2}(S))$
7. Commuting π with ⋈(or x): Suppose that the projection list is L = {A1, ..., An, B1, ..., Bm}, where A1, ..., An are attributes of R and B1, ..., Bm are attributes of S. If the join condition c involves only attributes in L, the two operations can be commuted as follows:
   - $\pi_L(R \bowtie_C S) = (\pi_{A1, \ldots, An}(R)) \bowtie_C (\pi_{B1, \ldots, Bm}(S))$
   - If the join condition C contains additional attributes not in L, these must be added to the projection list, and a final π operation is needed.

8. Commutativity of set operations: The set operations ∪ and ∩ are commutative but "–" is not.
9. Associativity of $\bowtie$, x, ∪, and ∩ : These four operations are individually associative; that is, if θ stands for any one of these four operations (throughout the expression), we have
   - $(R \theta S) \theta T = R \theta (S \theta T)$
10. Commuting σ with set operations: The σ operation commutes with ∪ , ∩ , and –. If θ stands for any one of these three operations, we have
    - $\sigma_c (R \theta S) = (\sigma_c (R)) \theta (\sigma_c (S))$

- The π operation commutes with ∪.

$$\pi_L (R \cup S) = (\pi_L (R)) \cup (\pi_L (S))$$

- Converting a (σ, x) sequence into $\bowtie$: If the condition c of a σ that follows a x Corresponds to a join condition, convert the (σ, x) sequence into a $\bowtie$ as follows:

$$(\sigma_C (R \times S)) = (R \bowtie_C S)$$

List the basic operations of following: Collection, Iterator, Set, Bag.

Iterator Operations

Operation	Description
Has Next	Check if there are more elements to iterate over.
Next	Get the next element in the iteration.
Remove	Remove the last element returned by the iterator.
Filter	Create an iterator that only returns elements matching a predicate.
Map	Apply a function to each element, producing a new iterator.
Reduce	Combine elements using a binary operator, producing a single result.

Operations common to Bag, Set and Collection

Operation	Description
Add	Add an element.
Remove	Remove a specific element.
Contains	Check for a specific element.
Size	Get the number of elements.

Operation	Description
Is Empty	Check if it is empty.
Iterator	Get an iterator over the elements .
Clear	Remove all elements.
Union	Combine two bags/sets/collections into one containing all elements.
Intersection	Get the common elements from two bags/sets/collections.
Difference	Get the elements from one bag/set/collection that are not in another.

How type constructors are created and used in SQL? Explain with an example.

In SQL, type constructors are used to define and create new data types. One common way to create user-defined types in SQL is through the use of composite types and domain types.

Composite Type: A composite type is a collection of fields, similar to a row or record.

Creating a Composite Type

```sql
CREATE TYPE address AS (
    street VARCHAR(100),
    city VARCHAR(50),
    state CHAR(2),
    zip_code CHAR(5)
);
```

Using the Composite Type

```sql
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    home_address address,
    work_address address
);
```

Inserting Data

```sql
INSERT INTO employees (name, home_address, work_address) VALUES (
    'John Doe',
    ('123 Main St', 'Anytown', 'CA', '12345'),
    ('456 Business Rd', 'Bigcity', 'NY', '67890')
);
```

Querying Data

```sql
SELECT
    name,
    home_address.street AS home_street,
    home_address.city AS home_city,
    work_address.street AS work_street,
    work_address.city AS work_city
FROM
    employees;
```

Domain Type: Domain is used to define constraints across multiple tables.

Creating a Domain

```
CREATE DOMAIN us_zip_code AS CHAR(5)
CHECK (VALUE ~ '^[0-9]{5}$');
```

Using the Domain

```
CREATE TABLE customers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    zip us_zip_code
);
```

Inserting Data

```
INSERT INTO customers (name, zip) VALUES ('Alice Smith', '54321');
```

How are horizontal and vertical partitioning of a relation specified?Explain.

Horizontal partitioning, also known as sharding, involves dividing a table into multiple smaller tables (partitions), where each partition contains a subset of the rows based on some partitioning criteria, often related to the values in one or more columns.

Specifying Horizontal Partitioning

1. **Range Partitioning:** Data is partitioned based on ranges of values in a column.

```
CREATE TABLE Sales (
    SaleID INT,
    SaleDate DATE,
    Amount DECIMAL(10, 2)
) PARTITION BY RANGE (SaleDate) (
    PARTITION p1 VALUES LESS THAN ('2022-01-01'),
    PARTITION p2 VALUES LESS THAN ('2023-01-01'),
    PARTITION p3 VALUES LESS THAN ('2024-01-01')
);
```

2. **List Partitioning:** Data is partitioned based on discrete values.

```sql
CREATE TABLE Employees (
    EmployeeID INT,
    Name VARCHAR(100),
    Department VARCHAR(50)
) PARTITION BY LIST (Department) (
    PARTITION p_sales VALUES IN ('Sales', 'Marketing'),
    PARTITION p_tech VALUES IN ('Engineering', 'IT'),
    PARTITION p_hr VALUES IN ('HR')
);
```

3. **Hash Partitioning:** Data is partitioned based on a hash function applied to a column.

```sql
CREATE TABLE Orders (
    OrderID INT,
    CustomerID INT,
    OrderDate DATE
) PARTITION BY HASH (CustomerID) PARTITIONS 4;
```

Vertical Partitioning: Vertical partitioning involves dividing a table into smaller tables based on columns. This method can be used to separate frequently accessed columns from less frequently accessed ones, optimizing query performance by reducing the amount of data read.

Specifying Vertical Partitioning

Example Table: Original table before partitioning.

```sql
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    Address VARCHAR(100),
    BirthDate DATE,
    Department VARCHAR(50),
    Salary DECIMAL(10, 2)
);
```

Partitioned Tables:

Table 1: Contains primary key and frequently accessed columns.

```sql
CREATE TABLE EmployeeMain (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    Department VARCHAR(50)
);
```

Table 2: Contains primary key and less frequently accessed columns.

```sql
CREATE TABLE EmployeeDetails (
    EmployeeID INT PRIMARY KEY,
    Address VARCHAR(100),
    BirthDate DATE,
    Salary DECIMAL(10, 2)
);
```

Discuss the different techniques for executing an equijoin of two tables located in different sites. What main factors affect the cost of data transfer?

1. **Naive Method**:

$$\pi_{R.*,S.*}(\sigma_{R.a=S.a}(R \times S))$$

2. **Semi-Join**:

$$\pi_a(R) \quad \rightarrow \quad S' = \sigma_{S.a \in \pi_a(R)}(S) \quad \rightarrow \quad R \bowtie_{R.a=S.a} S'$$

3. **Bloom Filter Join**:

$$\text{Create } BF(\pi_a(R)) \quad \rightarrow \quad S' = \sigma_{BF(S.a)}(S) \quad \rightarrow \quad R \bowtie_{R.a=S.a} S'$$

4. **Two-Phase Join**:

$$\text{Hash partition } R \text{ and } S \quad \rightarrow \quad \bigcup_{i=1}^{n} (R_i \bowtie_{R.a=S.a} S_i)$$

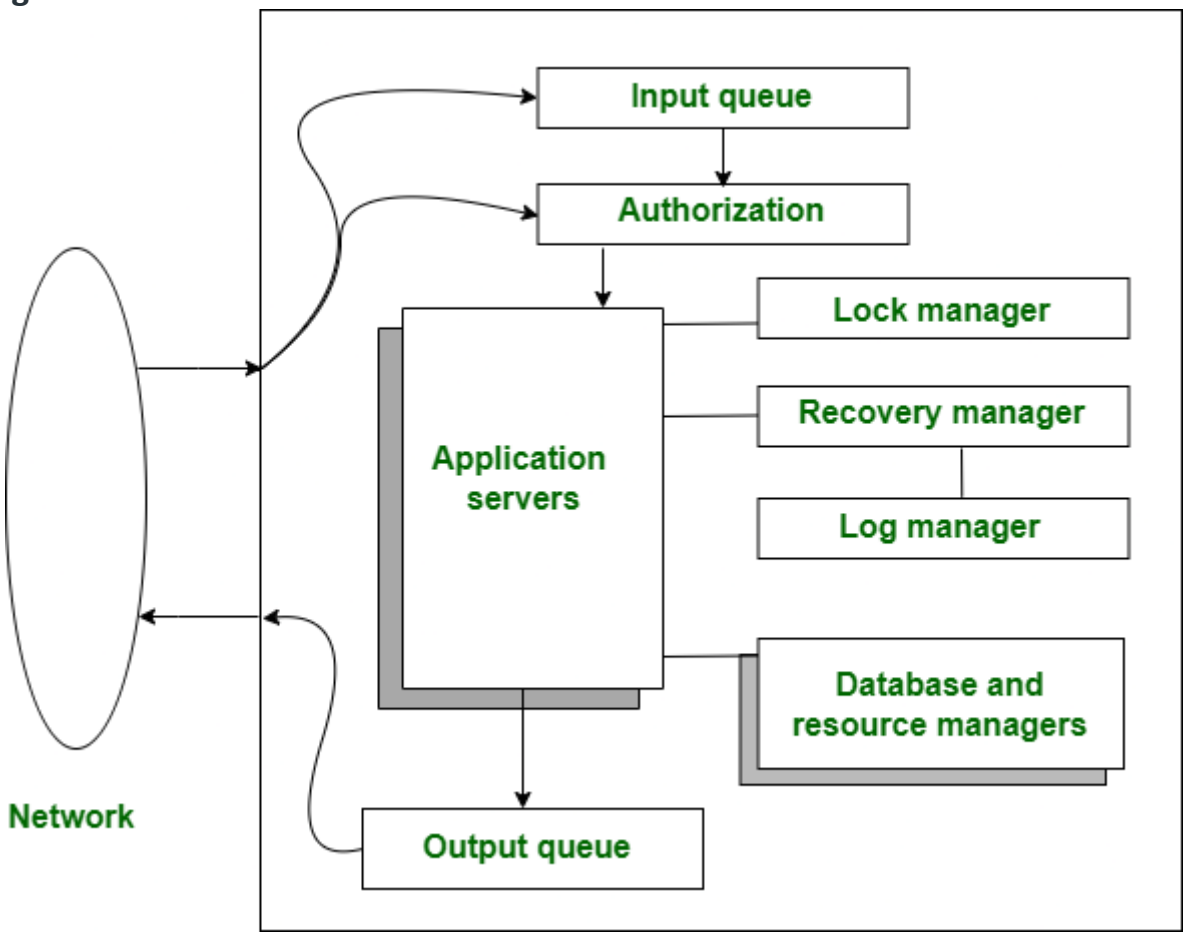
Factors Affecting the Cost of Data Transfer

- **Size of the Tables:** Larger tables result in higher data transfer costs.
- **Network Bandwidth:** Higher bandwidth results in efficient transfer of larger data.
- **Join Selectivity:** Higher selectivity means fewer tuples need to be transferred back and forth.
- **Fragmentation Strategy:** Efficient fragmentation reduces the amount of data that needs to be transferred.
- **Data Distribution:** Skewed distributions results in imbalanced data transfers and increased costs.

Explain the working of a TP monitor. What are various TP monitor architectures?

Transaction Processing Monitors acts as middlewares. Its main task is to support and handle interactions between applications on a variety of computer platforms.

Working:

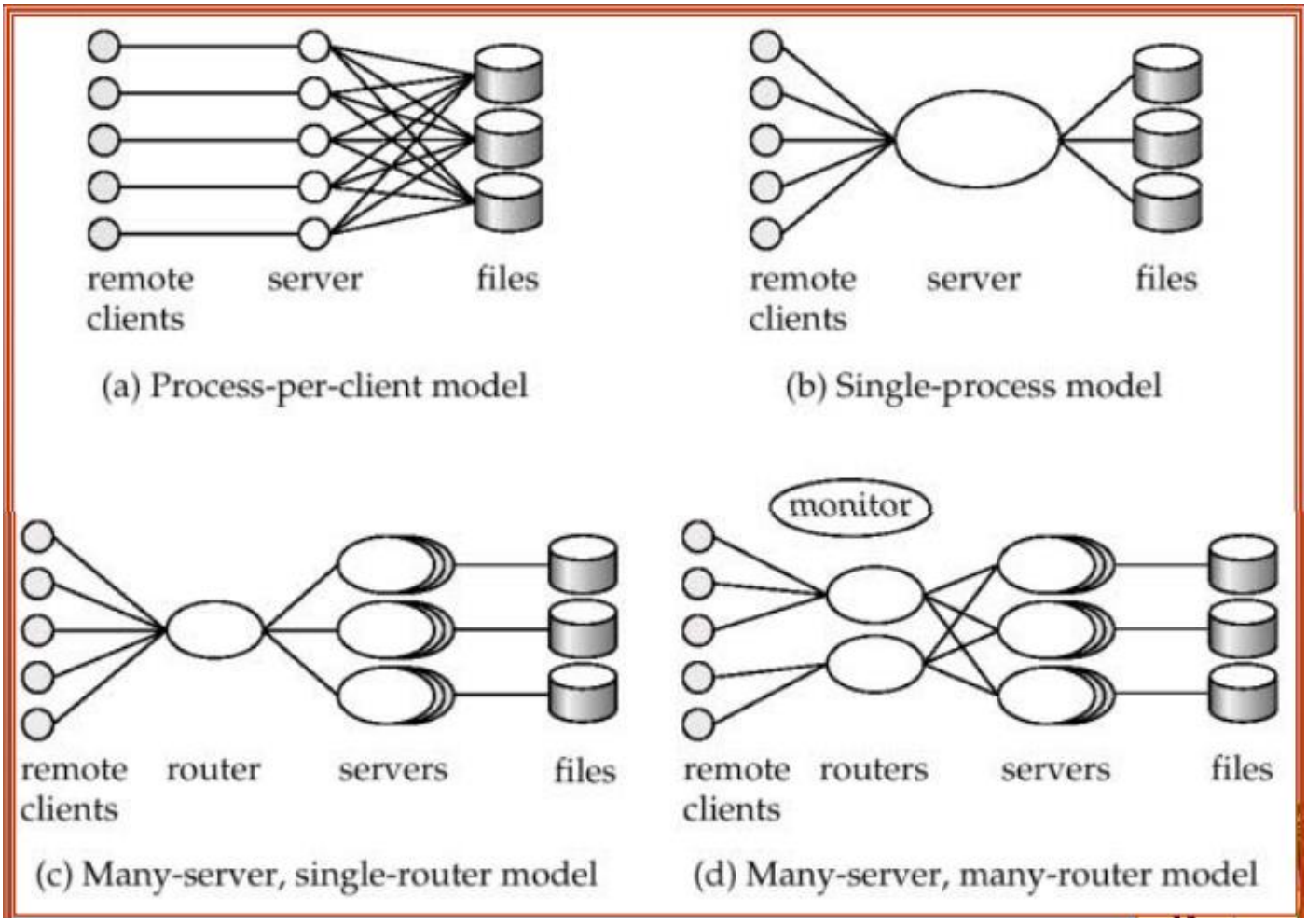


Working of TPM

- Incoming messages go from the queue manager or input queue as shown in the diagram above.
- The application server sends a confirmation message to the output queue as part of a transaction.
- Once the transaction completes, the TP monitor guarantees message is perfectly delivered.

TP Monitor Architectures:

- **Process per client model** - instead of individual login session per terminal, server process communicates with the terminal, handles authentication, and executes actions.
- **Single process model** - all remote terminals connect to a single server process.
- **Many-server single-router model** - multiple application server processes access a common database; clients communicate with the application through a single communication process that routes requests.
- **Many server many-router model** - multiple processes communicate with clients.



What are classification rules and how are decision trees associated with them

Classification rules are a set of IF-THEN rules that allow you to classify data. For example, 'IF the weather is sunny, THEN the person will play cricket.' The 'IF' part is the condition or rule, and the 'THEN' part is the decision accordingly. Decision trees can be seen as a set of classification rules. Each path from the root of the tree to a leaf represents a classification rule. The conditions along the path form the "if" part of the rule, and the class label at the leaf forms the "then" part.

Example:

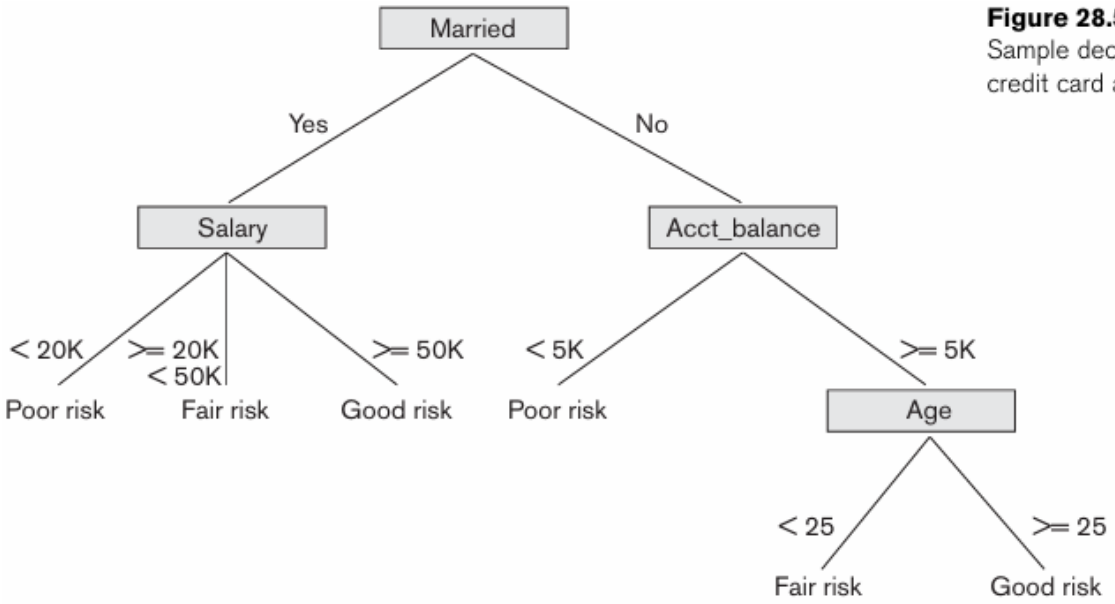


Figure 28.5
Sample decision tree for credit card applications.

Corresponding Classification Rules:

If Married=Yes and Salary<20K then Poor risk

If Married=Yes and 20K<=Salary<50K then Fair risk

If Married=Yes and Salary>=50K then Good risk

If Married=No and Acct_balance<5K then Poor risk

If Married=No and Acct_balance>=5K and Age<25 then Fair risk

If Married=No and Acct_balance>=5K and Age>=25 then Good risk

What is entropy and how it is used in building decision trees?

Entropy is a measure of uncertainty or impurity in a dataset. While building decision trees, entropy is used to determine how to split data at each node in the tree to achieve the most informative and purest subsets possible. The goal is to create nodes that partition the data into subsets with the lowest possible entropy, thereby creating the most homogenous child nodes.

Entropy is defined by the formula:

$$H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

where S is a set of data instances, p_i is the proportion of instances belonging to class i, and n is the number of classes.

Entropy ranges from 0 to 1:

0: If all the instances in the subset belong to a single class (pure).

1: If the instances are evenly distributed among all classes (maximal impurity).

What is the multidimensional data model? Explain roll up and drill down displays.

Multidimensional Data Model: It is used to show multiple dimensions of the data to users.

It represents data in the form of data cubes. Data cubes allow to model and view the data from many dimensions and perspectives. It is defined by dimensions and facts and is represented by a fact table.

Drill down: In drill-down operation, the less detailed data is converted into highly detailed data. It can be done by:

- Moving down in the concept hierarchy
- Adding a new dimension

Roll up: It is just opposite of the drill-down operation. It performs aggregation on the OLAP cube. It can be done by:

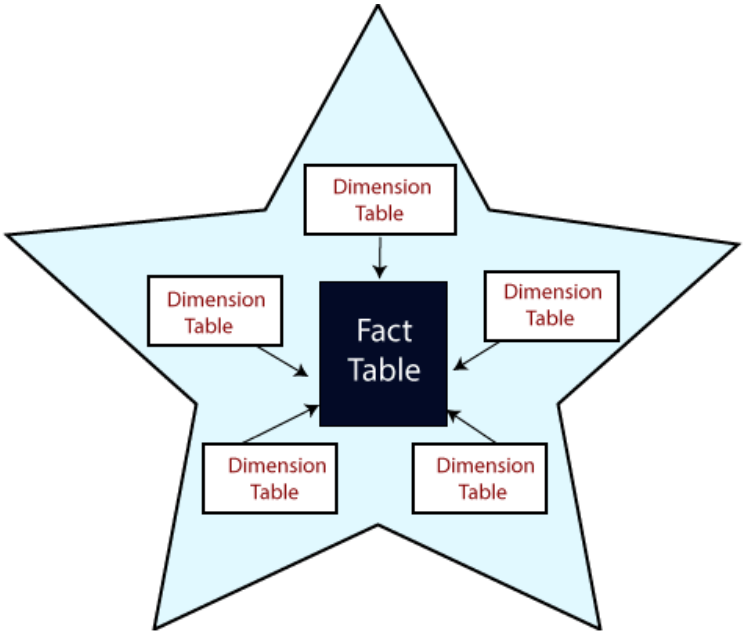
- Climbing up in the concept hierarchy
- Reducing the dimensions

Explain: Star schema, snowflake schema and fact constellation.

Star Schema: It has a central fact table that holds the primary data. The fact table is connected to multiple dimension tables, each representing different attributes related to the data in the fact table. The dimension tables are not directly connected to each other, creating a simple and easy-to-understand structure.

Features:

- **Simplicity**: Star schema is the simplest and most straightforward schema design, with fewer tables and relationships.
- **Denormalization**: Dimension tables in star schema are often denormalized, meaning they may contain redundant data to optimize query performance.

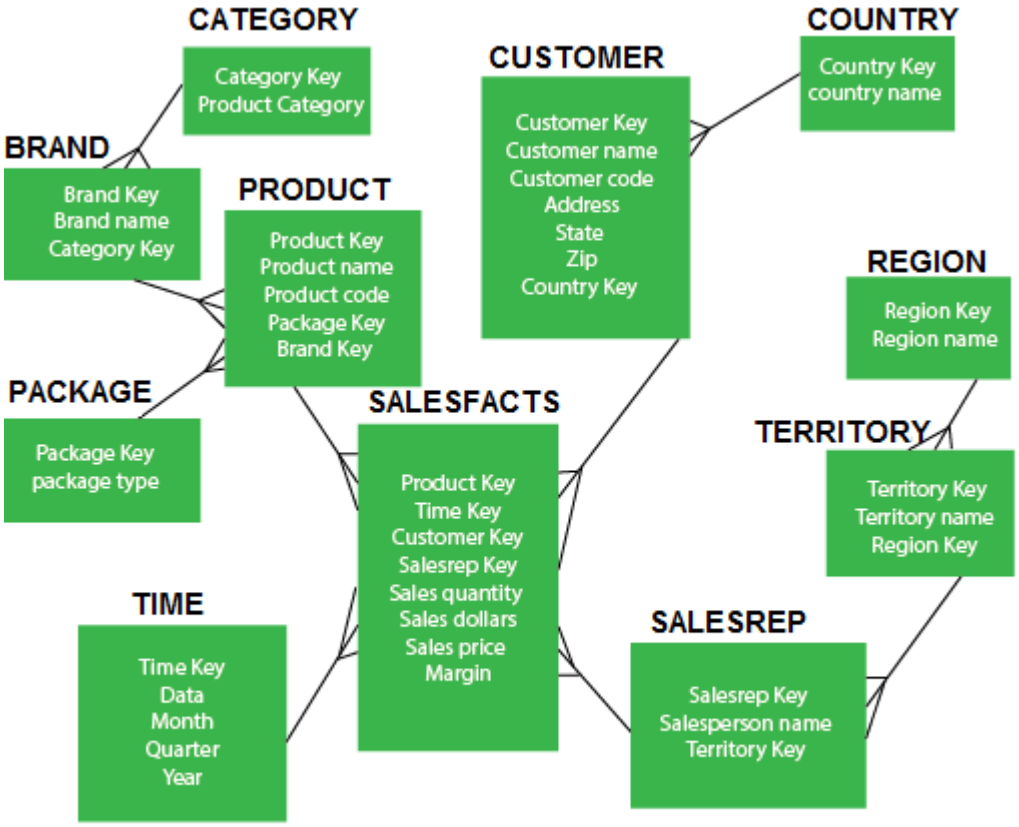


Star Schema

Snowflake Schema: The snowflake schema is an extension of the star schema, designed to further reduce data redundancy by normalizing the dimension tables. In a snowflake schema, dimension tables are broken down into multiple related sub-tables. This normalization creates a more complex structure with additional levels of relationships, reducing storage requirements but potentially increasing query complexity due to the need for additional joins.

Features:

- **Normalization:** Snowflake schema normalizes dimension tables, resulting in more tables and more complex relationships compared to the star schema.
- **Space Efficiency:** Snowflake schema requires less storage space for dimension data due to normalization.

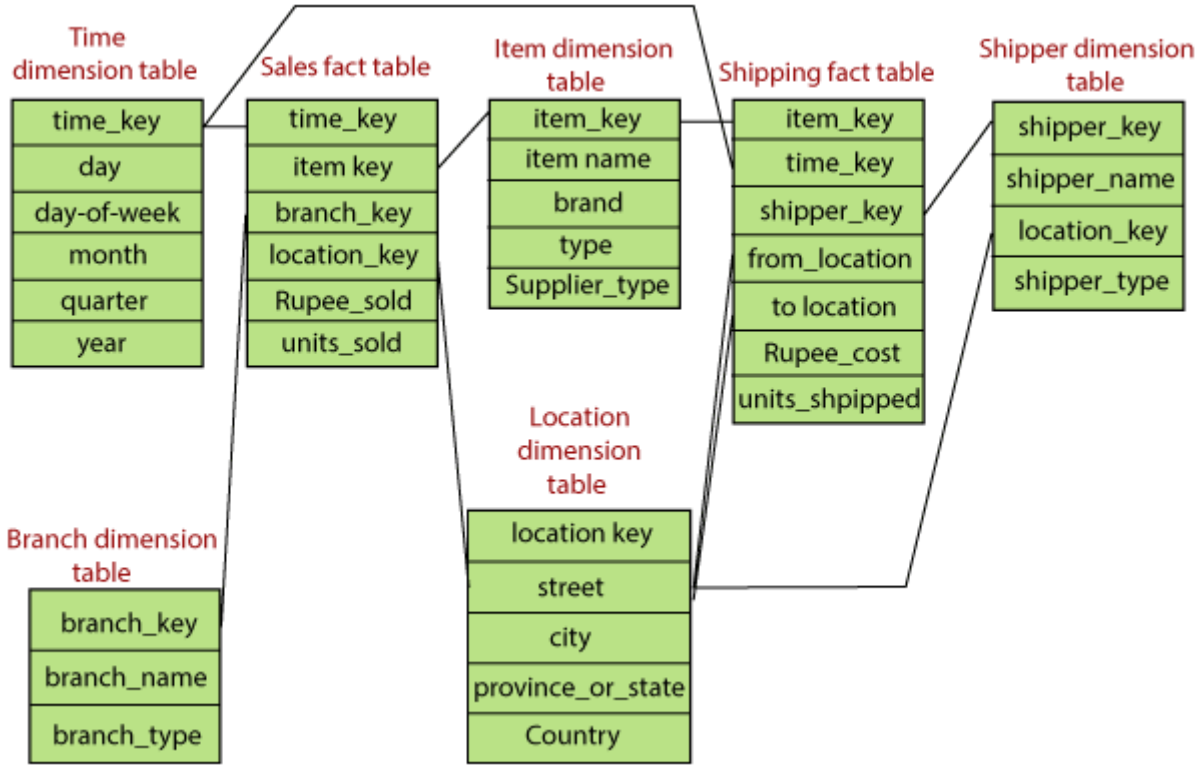


Snowflake Schema

Fact Constellation (Galaxy Schema): The fact constellation schema is a more complex design that involves multiple fact tables sharing dimension tables. It is used when there are multiple fact tables with different measures and each fact table is related to several common dimension tables.

Features:

- **Complexity:** Fact constellation schema is the most complex among the three designs, as it involves multiple interconnected star schemas.
- **Flexibility:** This schema design offers more flexibility in modeling complex and diverse business scenarios, allowing multiple fact tables to coexist and share dimensions.


Time dimension table
time_key
day
day-of-week
month
quarter
year
Sales fact table
time_key
item key
branch_key
location_key
Rupee_sold
units_sold
Item dimension table
item_key
item name
brand
type
Supplier_type
Shipping fact table
item_key
time_key
shipper_key
from_location
to location
Rupee_cost
units_shpipped
Shipper dimension table
shipper_key
shipper_name
location_key
shipper_type
Location dimension table
location key
street
city
province_or_state
Country
Branch dimension table
branch_key
branch_name
branch_type