Triggers

Triggers are similar to stored procedures in that they are self-contained units of SQL code. You don't explicitly call a trigger, as you do with a stored procedure. Instead, MySQL invokes the trigger automatically when a predefined event occurs. For example, suppose that you're setting up a database to support a CD retail business. The database includes two tables: the CDs table, which includes a list of CDs currently sold by the business, and the CDsPast table, which includes those CDs that have been sold in the past but are no longer carried. You can create a trigger so that, whenever a CD is deleted from the CDs table, it is automatically added to the CDsPast table. When you create the trigger, you assign it to the CDs table. The trigger includes an `INSERT` statement that adds the deleted CD to the CDsPast table when a `DELETE` event occurs on the CDs table.

As of the writing of this book, the MySQL product documentation did not include any specifics about how triggers will be created and implemented in MySQL. Most RDBMSs generally support three types of triggers: `insert`, `update`, and `delete`. Regardless of the type of trigger, you define it on a specific table, and when a related event occurs, the trigger is fired. For example, if you define an `insert` trigger on a table, MySQL fires the trigger when data is inserted in the table. When the trigger is fired, MySQL executes the SQL statements that are defined in the trigger. In the same way, MySQL fires `update` triggers when the table is updated or `delete` triggers when data is deleted from the table.

Triggers have been available in most RDBMSs for many years, so if you've worked with one of those other products, you already know how valuable triggers can be. They help to ensure that different tables in a database stay in sync and ensure that the proper action is taken if a particular event occurs. Without triggers, you have to perform many operations manually through SQL or your application.

TRIGGERS

Triggers exist in most DBMS to manage and monitor tables during insert, update or delete.

Purpose of Triggers

- Log changes on records
- Validation
- Creating backup or duplicates

CREATE TRIGGER Syntax

```
CREATE TRIGGER <trigger name>
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }
  ON <table name>
  FOR EACH ROW
     <triggered SQL statement>
```

Trigger Name

CREATE TRIGGER ***<trigger name>***
{ BEFORE | AFTER }
{ INSERT | UPDATE | DELETE }
ON <table name>
FOR EACH ROW
<triggered SQL statement>

Time

CREATE TRIGGER <trigger name>
{ BEFORE | AFTER }
{ INSERT | UPDATE | DELETE }
ON <table name>
FOR EACH ROW
<triggered SQL statement>

Types of SQL Triggers

- How many times should the trigger body execute when the triggering event takes place?
  - **Per statement:** the trigger body executes once for the triggering event. This is the default.
  - **For each row:** the trigger body executes once for each row affected by the triggering event.

- When the trigger can be fired
  - Relative to the execution of an SQL DML statement (before or after or instead of it)
  - Exactly in a situation depending on specific system resources (e.g. signal from system clock)

Event

CREATE TRIGGER <trigger name>
{ BEFORE | AFTER }
{ INSERT | UPDATE | DELETE }
ON <table name>
FOR EACH ROW
<triggered SQL statement>

Table

```
CREATE TRIGGER <trigger name>
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }
  ON <table name>
  FOR EACH ROW
     <triggered SQL statement>
```

Granularity

CREATE TRIGGER <trigger name>
{ BEFORE | AFTER }
{ INSERT | UPDATE | DELETE }
ON <table name>
FOR EACH ROW
<triggered SQL statement>

Creating a New Delimiter

- We can specify a new delimiter instead of semi-colon (;)
- We will most likely need a new Delimiter in creating triggers
- For this class, we will use // as the new delimiter
- Syntax: DELIMITER <New Delimiter>

Creating a New Delimiter

Examples:

```
mysql> DELIMITER //
```

```
mysql> DELIMITER <
```

```
mysql> DELIMITER >>
```

```
mysql> DELIMITER <<
```

Example of a Trigger Structure

```
DELIMITER //
CREATE TRIGGER tblEnrollment_bi
  BEFORE INSERT ON tblEnrollment
  FOR EACH ROW
  BEGIN
    SET @new = NEW.IDNo;
    SET @old = OLD.IDNo;
  END;//
```

Variables

- Variables are placeholders that hold a certain data.
- Variable Name Syntax: @<variable name>
- Examples of variable
  - @new
  - @old
  - @x

Assignment Statements

- <variables> = <values>
- Examples:

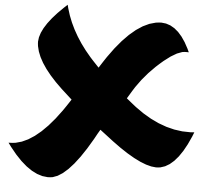
@X = 5; ← @X gets the value 5

@Y = @X; ← @Y gets the value variable @X

@X = @Y + 7; ← @X gets the value derived from @Y + 7 operation

Left variable always gets the right value

@X = 5 

5 = @X 

New and Old Columns

- OLD.<Column Name> - value of the column before it was updated.
- NEW.<Column Name> - value of the column after it was updated.

```
mysql > DELIMITER //
- > CREATE TRIGGER tblEnrollment_bi
- > BEFORE UPDATE ON tblEnrollment
- > FOR EACH ROW
- > BEGIN
- >    SET @new = NEW.IDNo;
- >    SET @old = OLD.IDNo;
- > END;//
```

- @new will get the value of column IDNO after it was updated
- @old will get the value of column IDNo before its was updated

Lets Practice!

Type the following and see what happens:

```
mysql> Select * from @old, @new; //
```

```
mysql> UPDATE tblEnrollment
-> SET IDNo = 55
- > WHERE EnrollmentNo = 1; //
```

```
mysql> Select * from @old, @new; //
```

Exercise:

Create a trigger that logs the following information to a table tblEnrollment_JNL every time a record is updated in tblEnrollment.

- User who modified the record
- Date when the record was modified
- The value of IDNo before the record was updated
- The value of IDNo after the record was updated

* **tblEnrollment_JNL(User, DateModified, Old_IDNo, New_IDNo)**

It should look like this:

```
mysql > CREATE TRIGGER tr_tblEnrollment
- > AFTER UPDATE ON tblEnrollment
- > FOR EACH ROW
- > BEGIN
- >    INSERT INTO tblEnrollment_JNL
- >    VALUES (user(), sysdate(), OLD.IDNo, NEW.IDNo);
- > END;//
```

DROP Trigger

```
mysql > DROP TRIGGER trigger_name [,…n]
```

Exercise

- Using tblEmployee of dbPersonnel (Chapter 12-13), create a trigger that will delete the records a user inserts in tblEmployee.

- EmpNo should be greater than 1002.

Event-Condition-Action (ECA)

- **E**vent occurs in databases
  - addition of new row, deletion of row by DBMS
- **C**onditions are checked
  - SQL condition
- **A**ctions are executed if conditions are satisfied
  - SQL + procedures
  - All data actions performed by the trigger execute within the same transaction in which the trigger fires,
  - Cannot contain transaction control statements (COMMIT, SAVEPOINT, ROLLBACK)

Database Triggers in SQL

- Not specified in SQL-92, but standardized in SQL3 **(SQL1999)**
- Available in most enterprise DBMSs (Oracle, IBM DB2, MS SQL server) and some public domain DBMSs (Postgres)
  - but not present in smaller desktop (Oracle Lite) and public domain DBMS (MySQL)
- Some vendor DBMS permit native extensions to SQL for specifying the triggers
  - e.g. PL/SQL in Oracle, Transact SQL in MS SQL Server
- Some DBMS also general purpose programming language instead of SQL
  - e.g. C/C++ in Poet, Java in Oracle, C#/VB in SQL Server
- Some DBMS extend the triggers beyond tables
  - for example also to views as in Oracle

Statement and Row Triggers

Example 1: Monitoring Statement Events

```
SQL> INSERT INTO dept (deptno, dname, loc)
  2  VALUES (50, 'EDUCATION', 'NEW YORK');
```

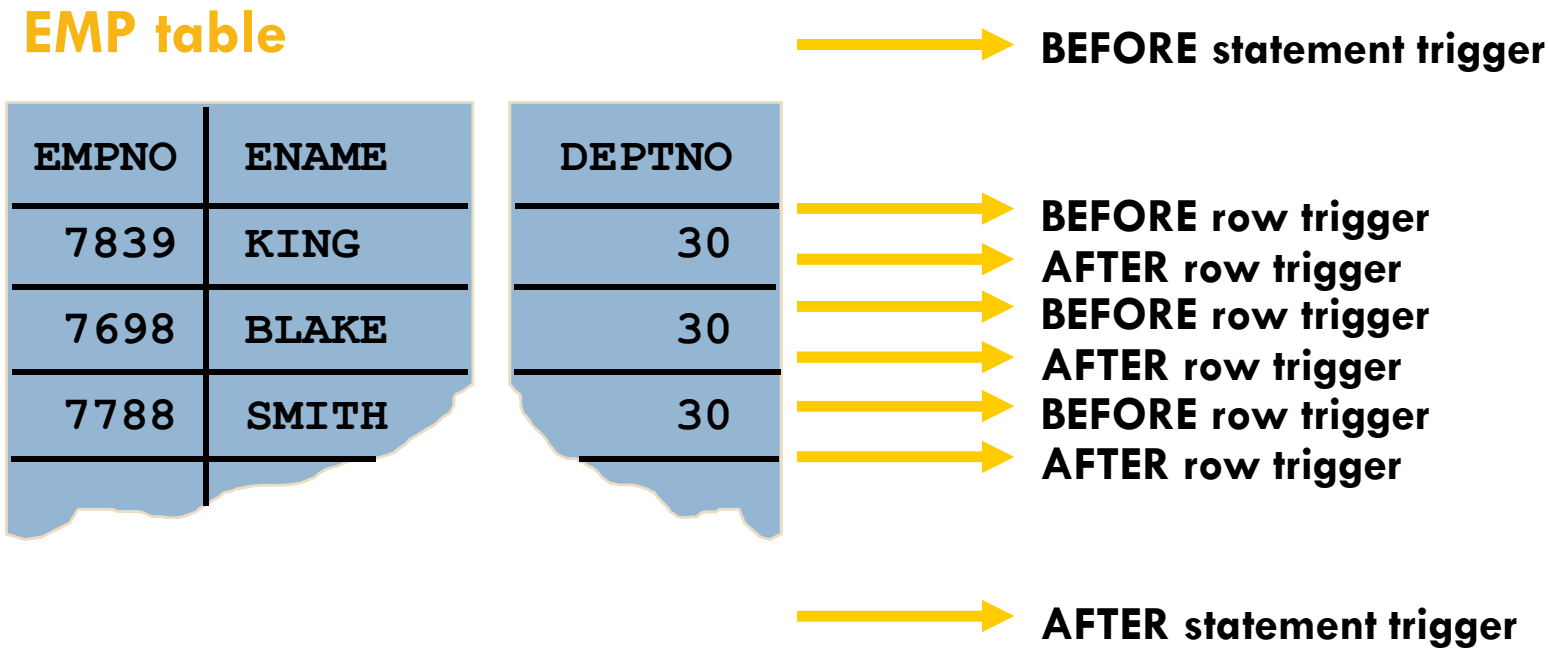
Execute only once even if multiple rows affected

Example 2: Monitoring Row Events

```
SQL> UPDATE emp
  2  SET sal = sal * 1.1
  3  WHERE deptno = 30;
```

Execute for each row of table affected by event

Firing Sequence of Database Triggers on Multiple Rows



Syntax for creating triggers in SQL

- **Trigger name** - unique within one database schema
- **Timing** - depends on the order of controlled events (before or after or instead of)
- **Triggering event** - event which fires the trigger **(E)**
- **Filtering condition** - checked when the triggering event occurs **(C)**
- **Target** - table (or view) against which the trigger is fired; they should be both created within the same schema
- **Trigger Parameters** - parameters used to denote the record columns; preceded by colon
  - **:new, :old for new and old versions of the values respectively**
- **Trigger action** - SQL statements, executed when the trigger fires; surrounded by **Begin** ... **End (A)**

Using Database Triggers

- **Auditing Table Operations**
  - each time a table is accessed auditing information is recorded against it
- **Tracking Record Value Changes**
  - each time a record value is changed the previous value is recorded
- **Protecting Database Referential Integrity:** if foreign key points to changing records
  - referential integrity must be maintained
- **Maintenance of Semantic Integrity**
  - **e.g.** when the factory is closed, all employees should become unemployed
- **Storing Derived Data**
  - **e.g.** the number of items in the trolley should correspond to the current session selection
- **Security Access Control**
  - **e.g.** checking user privileges when accessing sensitive information

Auditing Table Operations

USER_NAME	TABLE_NAME	COLUMN_NAME	INS	UPD	DEL
SCOTT	EMP		1	1	1
SCOTT	EMP	SAL		1	
JONES	EMP		0	0	1

... continuation

MAX_INS	MAX_UPD	MAX_DEL
5	5	5
	5	
5	0	1

Example: Counting Statement Execution

```
SQL>CREATE OR REPLACE TRIGGER audit_emp
  2 AFTER DELETE ON emp
  3 FOR EACH ROW
  4 BEGIN
  5       UPDATE audit_table SET del = del + 1
  6       WHERE user_name = USER
  7       AND table_name = 'EMP';
  7 END;
  8 /
```

Whenever an employee record is deleted from database, counter in an audit table registering the number of deleted rows for current user in system variable USER is incremented.

Example: Tracing Record Value Changes

USER_NAME	TIMESTAMP	ID	OLD_LAST_NAME	NEW_LAST_NAME
EGRAVINA	12-SEP-04	7950	NULL	HUTTON
NGREENBE	10-AUG-04	7844	MAGEE	TURNER

… continuation

OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
NULL	ANALYST	NULL	3500
CLERK	SALESMAN	1100	1100

Rules for Good SQL Practice

- **Rule 1:** Do not *change data* in the primary key, foreign key, or unique key columns of any table
- **Rule 2:** Do not *update records* in the same table you read during the same transaction
- **Rule 3:** Do not *aggregate* over the same table you are updating
- **Rule 4:** Do not *read data* from a table which is updated during the same transaction

Online Transaction Processing (OLTP)

- Immediate automated responses to the requests of users
- Handles multiple concurrent transactions from customers
- Fixed number of inputs per transaction
- Receiving user information, processing orders, and generating sales receipts (e.g., e-Commerce applications)

Operational Systems and BI

- Data from operational systems are useful inputs to BI applications.
  - Example: grocery checkout system data can be analyzed for spending patterns, effectiveness of sales promotions, or customer profiling.
- Informational systems—systems designed to support decision making based on stable point-in-time or historical data.
- Real-time analytical processing diminishes the performance of transaction processing.
  - Therefore, organizations replicate transactions on a second database server for analytical processing.

Operational vs. Informational Systems

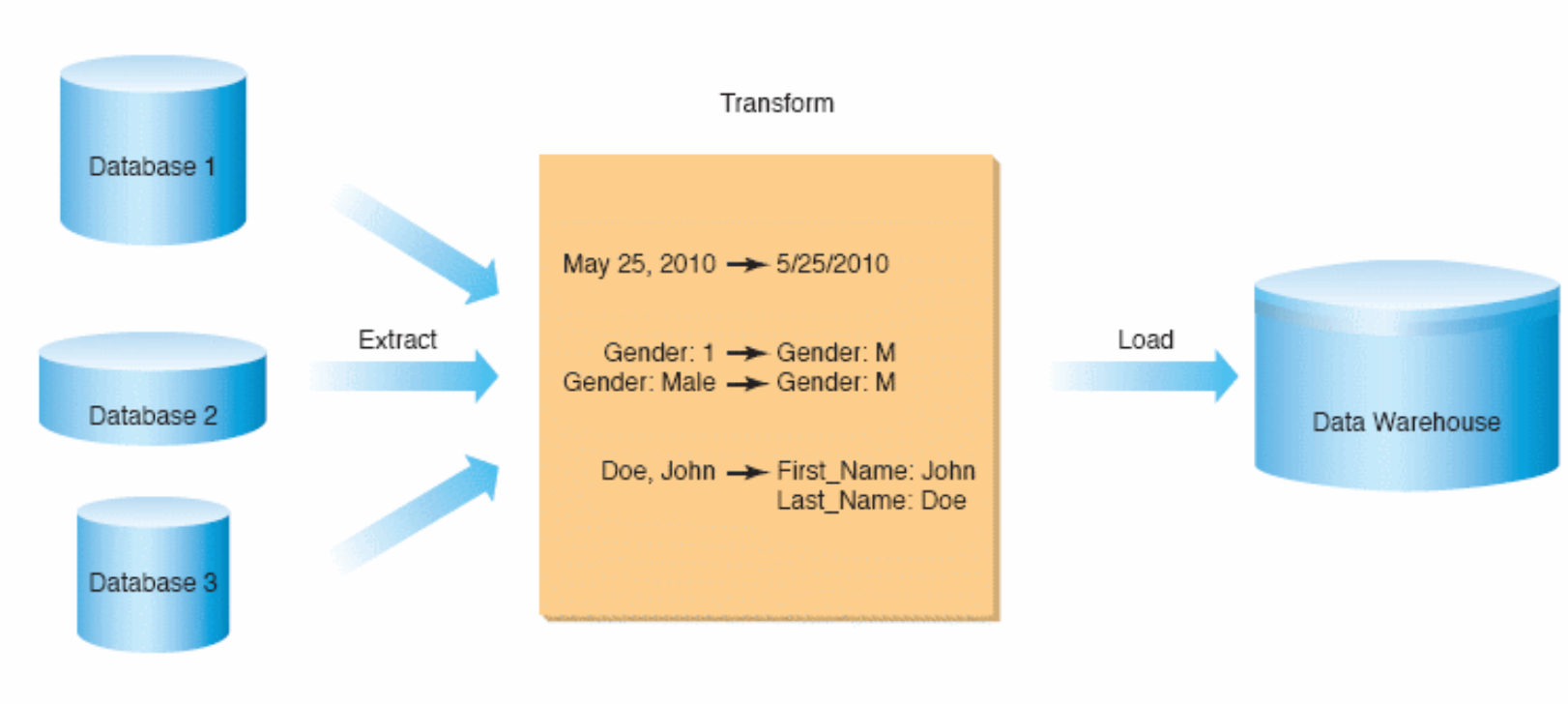
Characteristic	Operational System	Informational System
Primary purpose	Run the business on a current basis	Support managerial decision making
Type of data	Current representation of state of the business	Historical or point-in-time (snapshot)
Primary users	Online customers, clerks, salespersons, administrators	Managers, business analysts, and customers (checking status and history)
Scope of usage	Narrow and simple updates and queries	Broad and complex queries and analyses
Design goal	Performance	Ease of access and use

Data Warehouses

- Data warehouses integrate multiple databases and other information sources into a single repository.
- For direct querying, analysis, or processing
- Purpose: put key business information into the hands of decision makers.
- Take up hundreds of gigabytes (even terabytes) of data

Extraction, Transformation, and Loading (ETL)

- ETL is used to consolidate data from operational systems into a data warehouse.



Data Marts

- A data mart is a data warehouse that is limited in scope.
- Each data mart is customized for decision support of a particular end-user group.
- It is popular for small and medium-sized businesses and departments within larger organizations.
- Data marts can be deployed on less powerful hardware.

Online Analytical Processing (OLAP)

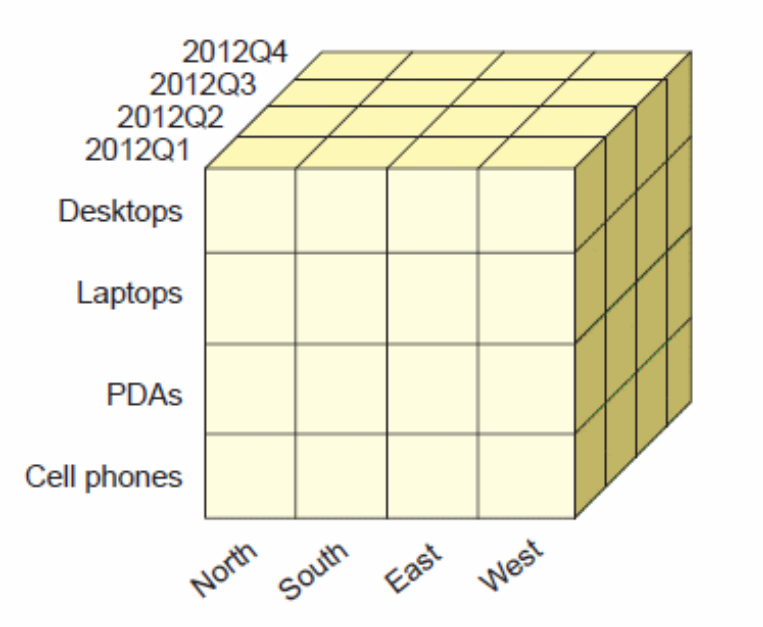
- Complex, multidimensional analyses of data beyond simple queries
- OLAP server —main OLAP component
- Key OLAP concepts:
  - Measures and dimensions
  - Cubes, slicing, and dicing
  - Data mining
  - Association discovery
  - Clustering and classification
  - Text mining and Web content mining
  - Web usage mining

Measures and Dimensions

- Measures (facts)—values or numbers to analyze.
  - Examples: sum of sales, number of orders placed
- Dimensions—groupings of data, providing a way to summarize the data.
  - Examples: region, time, product line
- Dimensions are organized as hierarchies (general-to-detailed).
  - Examples: year–month–day, state–county–city
- Drill-down—viewing measures at lower levels of hierarchy.
- Roll-up—viewing measures at higher levels of hierarchy.

Cubes

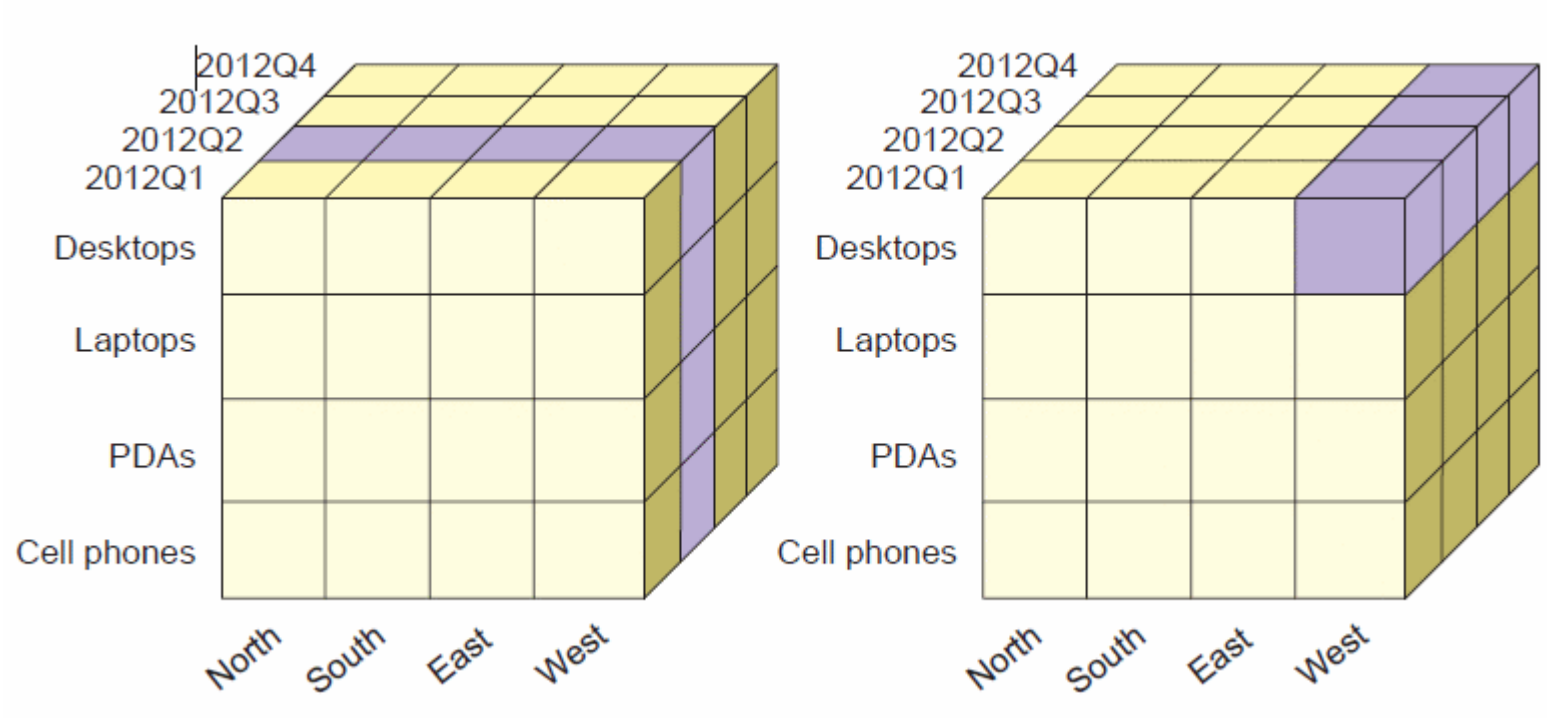
- Cube—an OLAP data structure organizing data via multiple dimensions.
- Cubes can have any number of dimensions.



A cube with three dimensions

Slicing and Dicing

- Slicing and dicing—analyzing the data on subsets of the dimensions

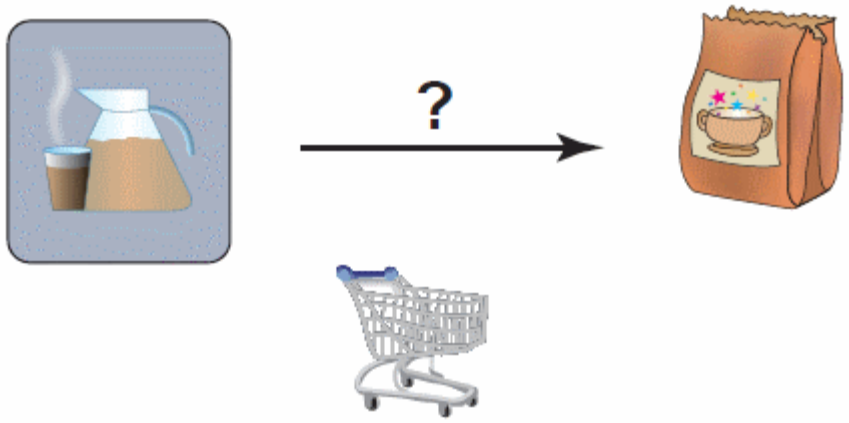


Data Mining

- Used for discovering "hidden" predictive relationships in the data
  - Patterns, trends, or rules
  - Example: identification of profitable customer segments or fraud detection
  - Any predictive models should be tested against "fresh" data.
- Data-mining algorithms are run against large data warehouses.
  - Data reduction helps to reduce the complexity 0f data and speed up analysis.

Association Discovery

- Association discovery—Technique used to find associations or correlations among sets of items.
  - Support and confidence indicate if findings are meaningful
- Sequence Discovery—Used to discover associations over time



Coffee → Sugar [Support 20%, Confidence 80%]

Clustering and Classification

- Clustering
  - Grouping of related records based on similar values for attributes
  - Groups are not known beforehand
    - Example: clustering frequent fliers based on segments flown

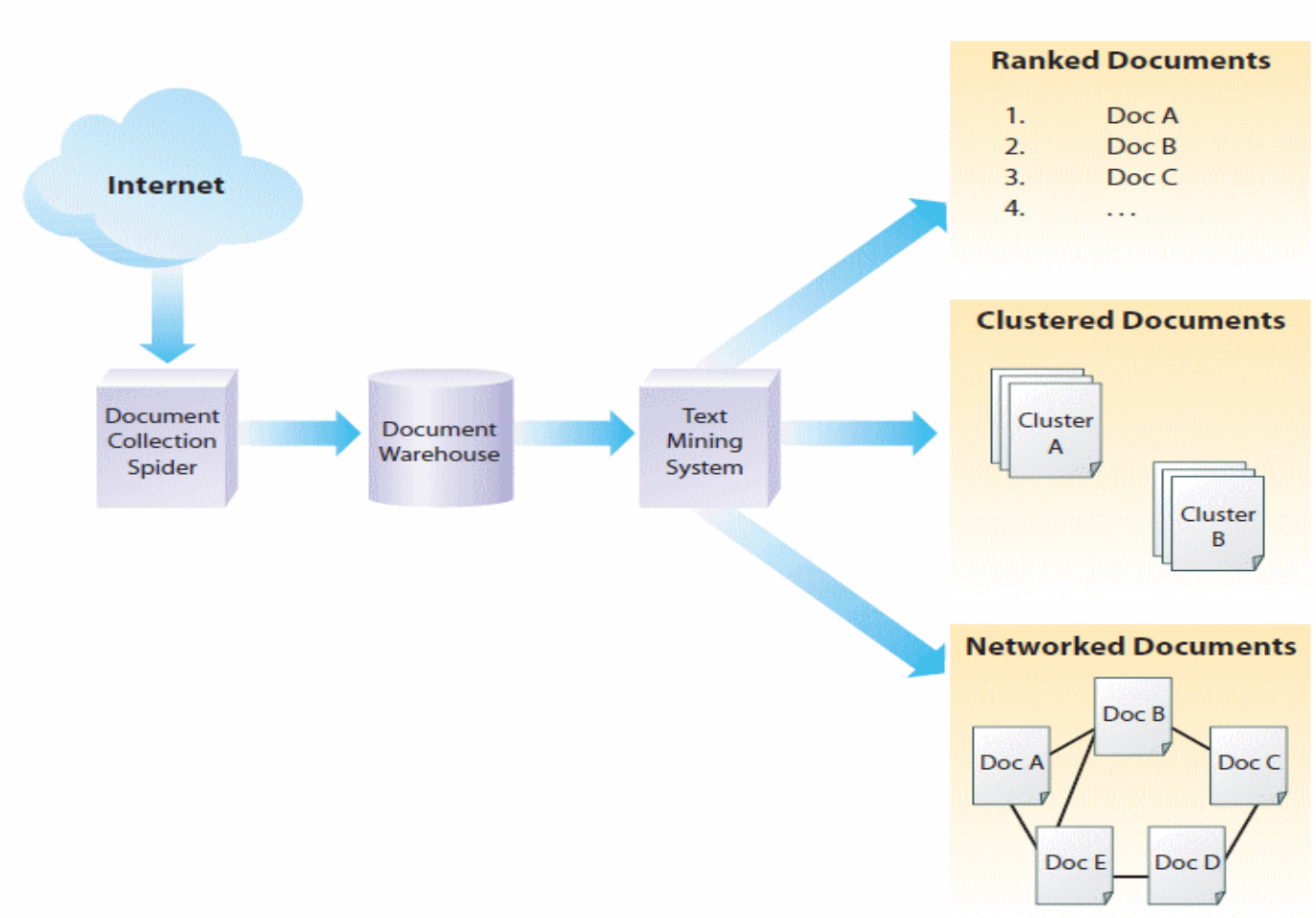
- Classification
  - Groups (classes) are known beforehand.
  - Example: A bank specifies classes of customers who differ in their risk categories (likelihood of defaulting on a loan).
  - Records are segmented into the different groups
    - Often using decision trees

Text and Web Content Mining

- Text mining—use of analytical techniques to extract information from textual documents.
  - Textual documents can include: Letters, e-mails, customer calls, internal communications, blog posts, wikis, Web. pages, marketing materials, patent filings, and so on
  - Text mining systems analyze a document's linguistic structures and key words.

- Web content mining—extract textual information from Web documents.
  - Web crawler searches sites and documents

Text mining the Internet



Textual Analysis Benefits

- Marketing—learn about customers' thoughts, feelings, and emotions.
- Operations—learn about product performance by analyzing service records or customer calls.
- Strategic decisions—gather competitive intelligence.
- Sales—learn about major accounts by analyzing news coverage.
- Human resources—monitor employee satisfaction or compliance to company policies (important for compliance with regulations such as the Sarbanes-Oxley Act).

Web Usage Mining

- Used by organizations such as Amazon.com
- Used to determine patterns in customers' usage data.
  - How users navigate through the site
  - How much time they spend on different pages
- Clickstream data—recording of the users' path through a Web site.
- Stickiness—a Web page's ability to attract and keep visitors.

Presenting Results