Module 4

Module 4 Study Guide and Deliverables

Readings:	Required Reading: • CB6 chapter 30 Recommended Reading: • Loney chapter 52
Assignments:	Assignment 4 due Wednesday, August 5 at 5:00 PM ET
Assessments:	Quiz 4 due Wednesday, August 5 at 5:00 PM ET
Term Project Note:	During this module you should complete much of the implementation of your project, including writing most of your research paper. You should provide evidence of progress to your instructor, who will review it and provide guidance.
Live Classrooms:	Supplementary Live Session Wednesday July 29, 8:00 PM - 10:00 PM ET Current week's assignment review and examples, Thursday, July 30, 8:00 PM - 9:00 PM ET Live office help, Saturday, August 1, 11:00 AM - 12:00 PM ET

Live office help, Monday, August 3,
8:00 PM - 8:45 PM ET

Lecture 4A - Semistructured Data and XML

Data and Structure

Structured data is data that conforms to a fixed schema. Data in a relational database is structured. For instance, in the hotel schema from your assignments, a hotel record is made up of a hotel number, a hotel name, a city, etc.

```
HOTEL
hotelno  Decimal(12)
hotel    Name Varchar(30)
city     Varchar(30)
```

If there is no value for an attribute, like city, a null value will "hold its place" in the structure. Every record in the hotel table has these same attributes in the same order. RDBMSes require a strict structure, and the data must conform to this structure. The hotel data in this example is structured.

Semi-structured data is either irregular data, or data in which the structure can change. Semi-structured data has some structure, but the schema can change rapidly and may not be consistent between records. Semi-structured data is also sometimes called "self-describing" or "schema-less." Big Data models often assume the data or structure will vary constantly, and so they have this flexibility built into their product. MongoDB, for instance, has a flexible schema. XML is possibly the most common way to model semi-structured data.

Unstructured data has no structure. It is data that can't be structured in an RDBMS format or in a semi-structured format, something like a video or text blog. Hadoop is one way to process and analyze unstructured data.

Extensible Markup Language (XML) and XQuery

XML is a text-based meta-language that has become the standard way to transfer data between businesses over the internet, and modern DBMS such as Oracle have XML data types to facilitate the storage, query and retrieval of XML data. Both HTML and XML are based on the SGML (Standard Generalized Markup Language). XML allows the document to define tags, where HTML has a fixed set of tags, so XML is more flexible. HTML is designed to support the presentation of data such as web pages. XML is designed to facilitate the communication of data between different organizations. An XML document is self-describing, meaning that a user can tell what the format of an XML document is by examining the document. This is different than a binary or other opaque document format, where the recipient must know the encoding to parse the document.

An XML document is organized as a tree data structure, with a single named node, or root, at the top and a hierarchy of named nodes beneath it - These named nodes can be either "elements" or "attributes"

XML consists of tags, like HTML. But unlike HTML, which constrains the user to a set of known element names, an XML document can define its own names for elements. In this example, the element is named “student”:

```
<student>Harry Jones</student>
```

Each starting tag has a matching end tag and the tags are case sensitive. Tags define the elements. Key/value pairs define the attributes within the element. In this example, “student” is the name of the element, and “studentID” is an attribute.

```
<student studentID="123">Harry Jones</student>
```

Data can be represented either as elements or as attributes. Best practices dictate that specifying an element is preferable if the data can be qualified further. For instance a “date” might be further broken down by month, day or year. In this case, it would be better to define date as an element. In the example given above, studentID will (probably) not be separated into different parts. If there’s any doubt about how the data will be used in the future, it’s best to define the data as an element.

Tags can also be nested in a containing element as the “instructor” tag is nested within the “course” tag:

```
<course>
<instructor>Professor Schudy</instructor>
</course>
```

The instructor tag starts and ends within the higher-level course tag.

Repeating groups are valid in XML. For instance, in this example, there are multiple students listed for one course:

```
<course>
<coursename>CS779</coursename>
<instructor>Professor Schudy</instructor>
<students>
<studentName>Alice Brown</studentName>
<studentName>Jane Jones</studentName>
<studentName>John Q. Public</studentName>
<studentName>Harry Smith</studentName>
</students>
</course>
```

An XML document may refer to a Document Type Definition (DTD) document that defines the valid tags and the relationships between the data elements. A DTD is like a data dictionary or schema for the XML documents of its type. An XML Schema Document (XSD) is an alternative representation for the structure of a class of XML documents. XSD supports the definition of domain constraints and other features that are useful in data interchange, so XSD is replacing DTD.

An XML document must be both well-formed and valid.

- To be well-formed, it must comply with certain rules, such as: The document starts with the XML declaration <XML version=”1.0” encoding=”UTF8”
- There is a root element that contains all the other elements. In other words, the document starts with a tag for the root element, like <course> and ends with its ending tag, in this case, </course>
- Elements cannot be overlapping.
- Attribute values must have quotes. They must be in the format name=”value”, like studentID=”123”.
- All start tags must have corresponding end tags.

To be valid, the document also must conform to the structural rules of XML. In other words, it must conform to a DTD or to an XML Schema. These documents define the structures of an XML document in a similar way as the DDL defines the structure of an RDBMS table.
A DTD lists all the elements, and their structure, the attributes they contain, how they’re nested, etc. There are several different types of DTD declarations:

- Entity type declarations
- Attribute list declarations
- Entity declarations

Element type declarations start with an exclamation mark and the keyword “ELEMENT”. The keyword #PCDATA indicates that the element contains text data that can be parsed. In other words, #PCDATA indicates that this is the text between the start and end tags. This is the way to declare an item element.

```
<!ELEMENT ITEM #PCDATA>
```

Elements can be nested within an element group. If we want to say that there can be multiple item elements per order, we would use an asterisk to indicate the cardinality. The DTD declaration would look like this:

```
<!ELEMENT ORDER (ITEM)*>
```

Note that ITEM would also be declared in the DTD as an element using PCDATA as above.

Attribute type declarations are similar but start with an exclamation mark followed by the keyword “ATTLIST”. Attribute declarations have three main parts, a name, a type and possibly a default value. So an attribute declaration for “student” above would look like this:

```
<!ATTLIST student studentID CDATA>
```

In this case, student is the element name, studentID is the attribute name, and CDATA is the attribute type. There are six different attribute types: CDATA, ID, IDREF, ENTITY, NMTOKEN or an enumerated type. The Connolly and Begg text describes these in more detail.

A newer alternative to defining the structure of an XML document is called *XML Schema*. XML Schema has some advantages over DTDs. An XML schema is in XML syntax. Declarations in XML Schema require types, which can be predefined, like primitive types, or they can be user-defined custom types. XML Schema also allows for namespaces.

An XML schema includes namespaces that identify the schema. The keywords, like “element” and “attribute” are defined in their own schema. The namespace definition starts with xmlns (xml namespace) followed by a prefix followed by the name of the namespace, like this:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

Namespaces are in the same form as a URL or hyperlink, but namespaces don't necessarily contain valid hyperlinks. The purpose of a namespace is to be able to qualify schema elements.

More than one namespace can be included in the same XML schema. Using different namespaces allows the schema to access different or remote schemas. Each namespace has a different prefix so a type or element defined in one schema will not be confused with a type or element with the same name in the imported schema. In the examples below, the tags are prefixed with "xs" which is the standard prefix given above for the XML Schema.

XML Schema has many built-in data types that resemble the primitive data types of other languages, like Java. These built-in types include boolean, string, decimal, float and date. The Connolly and Begg textbook describes these in more detail.

An XML Schema can also include user-defined types, which can be either a "simpleType" or a "complexType".

A simpleType is based on either a built-in type with a restriction, or extension, or it can also be based on another simpleType. A restriction can be either a limit on the values or give a regular expression that the data must match.

For instance, you're the instructor and you want to record a student grade. You want the format to be something like 100.0, or 99.8. You can do this by restricting the decimal type with a pattern, like this:

```
<xs:simpleType name="grade">
<xs:restriction base="xs:decimal">
<xs:pattern value="[0-9]{3}.[0-9]">
</xs:restriction>
</xs:simpleType>
```

In the example above, the "grade" must match the pattern of 3 integers, then a decimal point, followed by one number to the right of the decimal point.

A complexType can be a collection of simpleTypes or elements or both. It is possible to define a specific sequence of elements. For instance, if you wanted to ensure that last name came before first name, you could create a nameType as a complexType and define it this way:

```
<xs:complexType name="NameType">
<xs:sequence>
<xs:element name="lastName" type="xs:string"/>
<xs:element name="firstName" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

You could then use this NameType when you define other elements, like so:

```
<xs:element name=”shipToName” type=”NameType”></xs:element>
<xs:element name=”billToName” type=”NameType”></xs:element>
```

While XML documents are readable and easily parsed, they are not exactly user friendly, so they are not normally presented to users as XML. The XML standards provide a separate way to define how XML data is to be presented, in the form of the Extensible Style Language (XSL). XSL includes the Extensible Style Language Transformations (XSLT) and XSL style sheets. XSLT is a flexible language for specifying transformations on XML documents, including data extraction and mapping to other documents, including HTML or XML. XSL style sheets specify the formatting to be applied to XML elements when they are presented to the user. XSLT documents thus specify XML processing operations and XSL style sheets specify how XML data is to be presented.

Note

In addition to our textbooks, MSDN has a very thorough and well-presented discussion of XML at http://msdn.microsoft.com/en-us/library/aa286548.aspx

Web Services

A web service is a communication between a client and server. It is comprised of reusable software components that are exchanged between client and server. Microsoft has an excellent definition of web services at http://msdn.microsoft.com/en-us/library/ms950421.aspx

Web services extend the World Wide Web infrastructure to provide the means for software to connect to other software applications. Applications access Web services via ubiquitous Web protocols and data formats such as HTTP, XML, and SOAP, with no need to worry about how each Web service is implemented. Web services combine the best aspects of component-based development and the Web.

XML is one of the most commonly used transfer mediums for these services. It is platform-independent and open standard, both of which make it ideal for using with web services.

The Google Map API is one example of a web service. The client supplies a location to the service, and the Google map web service finds the latitude and longitude for that location, and returns a map.

The server that makes the service available “publishes” the service, and the client application that accesses the service “consumes” the service.

XPath and XQuery

The World Wide Web Consortium (W3C) has defined two closely related query languages, XPath and XQuery, designed to operate against data represented in XML. The standards documents for these languages can be found at http://www.w3.org/TR/xquery/. To understand querying XML documents we first need to understand the structural relationship between relational tables and XML trees. XML documents are tree-structured, which is a generalization of relational data representation. This is illustrated in the following examples, where the structure of an XML document is represented first, and then a relational table storing the same data is represented.

Example XML to Represent Staff Information

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<?xml:stylesheet type="text/xsl" href="staff_list.xsl" ?>
<!DOCTYPE STAFFLIST SYSTEM "staff_list.dtd">
<STAFFLIST>
  <STAFF branchNo="B005">
    <STAFFNO>SL21</STAFFNO>
      <NAME>
        <FNAME>John</FNAME><LNAME>White</LNAME>
      </NAME>
    <POSITION>Manager</POSITION>
    <DOB>1945-10-01</DOB>
    <SALARY>30000</SALARY>
  </STAFF>
  <STAFF branchNo="B003">
    <STAFFNO>SG37</STAFFNO>
      <NAME>
        <FNAME>Ann</FNAME><LNAME>Beech</LNAME>
      </NAME>
    <POSITION>Assistant</POSITION>
    <SALARY>12000</SALARY>
  </STAFF>
</STAFFLIST>
```

After Connolly & Begg (2015), p. 1092

Nodes of the XML Document Represented as Tuples of a Relation (after Connolly & Begg (2015), p. 1151)

nodeID	nodeType	nodeName	nodeData	parentID	rootID
0	Document	STAFFLIST			0
1	Element	STAFFLIST		0	0
2	Element	STAFF		1	0
3	Element	STAFFNO		2	0
4	Text		SL21	3	0
5	Element	NAME		4	0
6	Element	FNAME		5	0

7	Text		John	6	0
8	Element	LNAME		7	0
9	Text		White	8	0

Example Tuples in a (Denormalized) Index (after Connolly & Begg (2015), p. 1151)

path	nodeID	parentID
/STAFFLIST	1	0
STAFFLIST	1	0
STAFFLIST/STAFF	2	1
STAFF	2	1
/STAFFLIST/STAFF/NAME	5	2
STAFFLIST/STAFF/NAME	5	2
STAFF/NAME	5	2
NAME	5	2

XQuery is a generalization of SQL to tree-structured documents. XQuery is based on the W3C standard XPath, and the simplest kind of XQuery is an XPath expression such as.

```
document("family_recipes.xml")//recipe[title="Chard with
garlic"]//ingredient[@amount]
```

The result of this XQuery is a list of XML fragments, each rooted in an `ingredient` element. The order in the list is the same as the order in the XML document family_recipies.xml. Recall that the order of the returned rows is indeterminate in SQL, so XQuery is different, because the XML specifies an order for child nodes in the tree, and XQuery respects this order.

An XQuery may be an XML expression, in which case the expression evaluates to itself, so this is the way to specify XML literals in XQuery. XQuery's XML expressions may also include variables to be bound from the context, such as

```
<facilitator empid="{$id}">
<name>{$name}</name>
{$section}
```

```
<deptno>{$deptno}</deptno>
</facilitator>
```

Here the variables `$id, $name, $section,` and `$deptno` must be bound to XML fragments. Between the braces can be any XQuery expression.

FLWOR Expressions

The main syntax of XQuery is the FLWOR (pronounced “flower”) expression. FLWOR stands for “For-Let-Where-Order-Return.” FLWOR is a generalization of the Select-From-Where-Having syntax of SQL. Consider an example:

```
for $c in document("courses.xml")//coursecode
let $s:=document("sections.xml")//section[coursecode=$c]
where count($s) >= 7
order by $c ascending
return
<big-classes>
{ $c,
<numsections>{count($s)}</numsections>
}
</big-classes>
```

The following table describes what each of the FLWOR clauses does.

Clause	What it does
`for`	Generates an ordered list of bindings of `coursecode` values to `$c`.
`let`	Associates with each of these bindings a further binding of the list of sections elements with that course code to `$s`. The result is an ordered list of duples of bindings of (`$c,$s`).
`where`	Filters that list to retain only the desired bindings where there are more than seven sections in the course.
`order`	Sorts that list by the course name.
`return`	Constructs for each tuple a resulting value consisting of the course code and the number of sections.

The result set of this query is a list of courses with at least 7 sections, sorted by course names. The `for` and `let` clauses may be used any number of times in any order, but only one `where` clause is allowed, though that `where` clause can have many conditions, as in SQL.

The `for` clause generates a list of bindings, while `let` generates only a single binding. For example, the clause

```
for $c in /program/courses
```

generates a list of bindings of $c to each course element in the program, but

```
let $c := /program/courses
```

generates a single binding of $c to the list of courses in the program.

XQuery is a full-featured request language that is more flexible than SQL. It allows joining of documents in the same way that SQL supports joining of tables using multiple for clauses and if-then-else in the return clause. XQuery has where clause constructs *some-in-satisfies*, which is true if a predicate is true for any elements of the list, and *every-in-satisfies*, which is true if the predicate is true for all elements in the list. XQuery is a rather large language, with hundreds of built-in functions, which can be found at http://www.w3.org/TR/xpath-functions.

XML Databases

There are two general kinds of databases that support XQuery

- ***Native XML databases*** which are built from the ground up to support XML. These are generally startup efforts with weak security, transactions, scalability, recovery, etc., though they are maturing rapidly.
- ***XML extensions to relational and object-relational databases***. Most commercial RDBMS and ORDBMS support an XML data type and at least a subset of XQuery. Because these extensions combine relational technology they can be a little more complicated, but with care they can perform well, and they have all of the desirable properties of the mature databases of which they are extensions. The ability to support relational and XQuery operations in the same database improves flexibility and scalable performance.

An XML document can be stored in a relational database by shredding it. In shredding there is one table for each element type, a unique id for each occurrence of each element with the ids of siblings ordered as in the XML document. There are explicit foreign key references from children to the parent elements to define the tree structure.

Native XML Databases

XML databases can either be based on a *data-centric* or *document-centric* model. In a data-centric model, the data is extracted from the XML and stored in a structured format. XML is used as the transfer mechanism, but since the data has been extracted, any other form of transfer format could be used.

In a document-centric model, or Native XML Database, the data is stored as an XML document. These databases retain the structure of the XML document. The document might be stored either as text, like in a BLOB in a relational database, or it could be stored in a “model-based format”. The model-based format might contain entities like “Elements”, “Attributes”, etc. that mimic the structure of the XML. The focus of the document-centric model is on the XML document itself, rather than on the business logic behind the data.

An NXD is designed to store the document, and easily reconstruct the original document. These documents can be stored with any kind of underlying physical database, including relational or object-oriented databases, or its own proprietary structure. NXDs use XML commands, like XPath and XQuery. (Staken, 2001)

Vendors of Native XML Databases include Oracle (Berkely DB XML), EMC (Documentum XDB), and there are some popular open-source versions as well, Sedna, and eXist. This link has a more complete list of Native XML Databases, http://www.rpbourret.com/xml/XMLDatabaseProds.htm#native. (Bourret, 2010).

XML APIs

To easily work with XML from programming languages we need support from portable Application Programming Interfaces (APIs) and libraries. The XML APIs DOM and SAX are supported by many programming languages. The DOM derivative JDOM is supported in Java.

XML APIs provide interface programs and libraries to parse and process XML. They are based either on the document-object model (DOM), which builds a tree-based representation of the XML, or on the SAX (Simple API for XML) model, which is event-driven. DOM APIs are useful for manipulating the actual XML structure, while the SAX APIs are useful for parsing the data.

A SAX model is event-driven, so as the program processes the document (reading from top to bottom), it handles each "event". "START_DOCUMENT" is one event, for instance, that starts the parsing. The "START_ELEMENT" event causes the event handler to process an element, and to continue processing it until an "END_ELEMENT" event occurs when the closing tag is encountered In this example, for instance, the parser would record several events.

```
<coursename>CS779</coursename>
```

The events would include:

- `START_ELEMENT` as the parser read `<coursename>`. Getting the *Localname* value would return "coursename".
- `CHARACTERS` is the next event. This returns the text for that element. In this case "CS779".
- `END_ELEMENT` as the parser read `</coursename>`. Getting *Localname* here would again return the value "coursename".

SAX interfaces process the document from beginning to end and, as one event completes, it is no longer referenced by the parser. A DOM parser, on the other hand, recreates the structure of the XML "tree" starting at the root. So it is easier to reference other elements in the XML.

XML Services and Standards

XML is the standard language for communications with and between web services. Most industries have published standard XML formats that are used by these services and by their clients. The machine and language independence of XML, and its self-describing character, its extensibility, its standardization, and other advantages have led to the rapid adoption of XML for data interchange between enterprises. Most industry standards bodies have now adopted XML standards for communication within their industries. The future promises additional support for XML in databases, and perhaps the emergence of new databases and database extensions that further extend the reach of XML and XQuery into databases.

Lecture 4B - Introduction to Business Intelligence

Business Intelligence Terminology

The term *business intelligence* covers a range of information technology activities that support the understanding and management of the enterprise. Over the last forty years a substantial technology and vocabulary has developed for business intelligence. Many of the most important business intelligence terms are defined in the following table.

Term	Definition
DSS	A *Decision Support System* is any system that is used to support management or operational decision-making, including a warehouse, mart, or data analysis system
Data Warehouse	An integrated, subject-oriented, time-variant, nonvolatile database that provides support for decision making
Data Mart	A DSS that supports a functional area, region or other part of an enterprise
Business intelligence	The technologies and their applications for decision support, including data warehousing and data mining
ETL	*Extract, Transform and Load;* the processing that collects data from operational databases and other data sources, cleanses it, transforms it, summarizes it, and loads it into a data warehouse
OLAP	*Online Analytic Processing;* any decision support processing that responds to the user on a time scale that permits timely repeated queries. OLAP is distinct from overnight and other offline analytic processing
ROLAP	*Relational Online Analytic Processing;* an OLAP system built on a relational database management system
Dimensional Schema	A relational database design pattern with fact and dimension tables
Star Schema	Relational database design for data warehouses with one fact table and usually many dimension tables
Snowflake Schema	A star schema with additional tables supporting the dimensions, particularly scale hierarchies
Constellation Schema	A star or snowflake schema with multiple fact tables
MOLAP	*Multidimensional Online Analytic Processing;* OLAP based on a multidimensional database management system

Multidimensional DBMS (MDBMS)	A DBMS based on techniques such as sparse array representations of data cubes. Distinct from RDBMS
Trickle feed	An emerging technique where DSS databases are updated nearly continuously from OLTP databases and other real-time data feeds

Differences between DSS Databases and OLTP Databases

The following table summarizes the many differences between online transaction processing databases and decision support databases.

Characteristic	Operational Databases	DSS Databases
Role in enterprise	Value chain operations	Decision support
Organization	Organized by application	Organized by subject
Request complexity	Simple transactions	Complex, free-form queries
Request rate	May be very high	Relatively few
DB rows accessed for each request	Few rows accessed for each request	Billions of rows may be accessed for each request
Request latency	Requests usually completed within milliseconds	Requests may run for many minutes
Data currency	Dynamic, real-time data	Usually data for pre-defined past periods such as days
Optimization criteria	Optimized for insert/update/delete	Optimized for select
Normalization	Usually fully normalized	Data may be stored redundantly at multiple levels of aggregation
Time span	Weeks to a year of data	May have decades of data
Database type	Usually relational SQL databases	May be relational, multidimensional, or both
Indexing	Lightly indexed to minimize update cost	Heavily indexed to speed ad hoc queries

Query types	Transactional updates when in use	Only queries when in use
DB size	Megabytes to a few gigabytes	Gigabytes to hundreds of terabytes
Data deletion	Data have a finite lifetime	Old data are never deleted

In the early days of databases decision support queries were run overnight against operational databases. These queries were usually crafted by the technical specialists in the nascent information technologies departments. To understand why these queries ran overnight, consider what a simple query might look like running against simple operational sales database.

Simple Operational Sales Database Query

```
SELECT Category_Desc,
DATEPART(year, oh.Order_Date) "Year",
DATEPART(month,oh.Order_Date) "Month",
sum(DISTINCT(1+Tax_Rate)*(Units_Sold*ol.Price)) TotSales
FROM Order_Header oh, Order_Line ol,
Store s, Product p, SubCategory sc,
Category c
WHERE oh.Order_Number = ol.Order_Number
and oh.Store_Number = s.Store_Number
and ol.product_ID = p.product_ID
and p.product_ID = sc.product_ID
and sc.subcategory_ID = c.subcategory_ID
and Category_Desc = 'beverage'
and DATEPART(year, oh.Order_Date) IN (2004, 2005)
and DATEPART(month,oh.Order_Date) = 2
GROUP BY Category_Desc, DATEPART(year,oh.Order_Date),
DATEPART(month,oh.Order_Date)
```

Note: This query uses SQL Server syntax.

This query joins five tables, uses two function calls in the WHERE clause, and two in the GROUP BY clause. As we learned in week 2, without special features such as function indexes there is no efficient query plan for this query, and if the OLTP database against which it runs is of any size, it will take a long time to complete.

Dimensional Star Schema

Because management needed the ability to get the answers to questions more quickly, and because some queries just didn't run fast enough, people began looking for better database designs to support ad hoc decision support queries. Over time the dimensional design principles were developed that allow relational databases to efficiently support a wide range of ad hoc queries, even with very large databases. The following figure illustrates a dimensional star schema for the same problem.

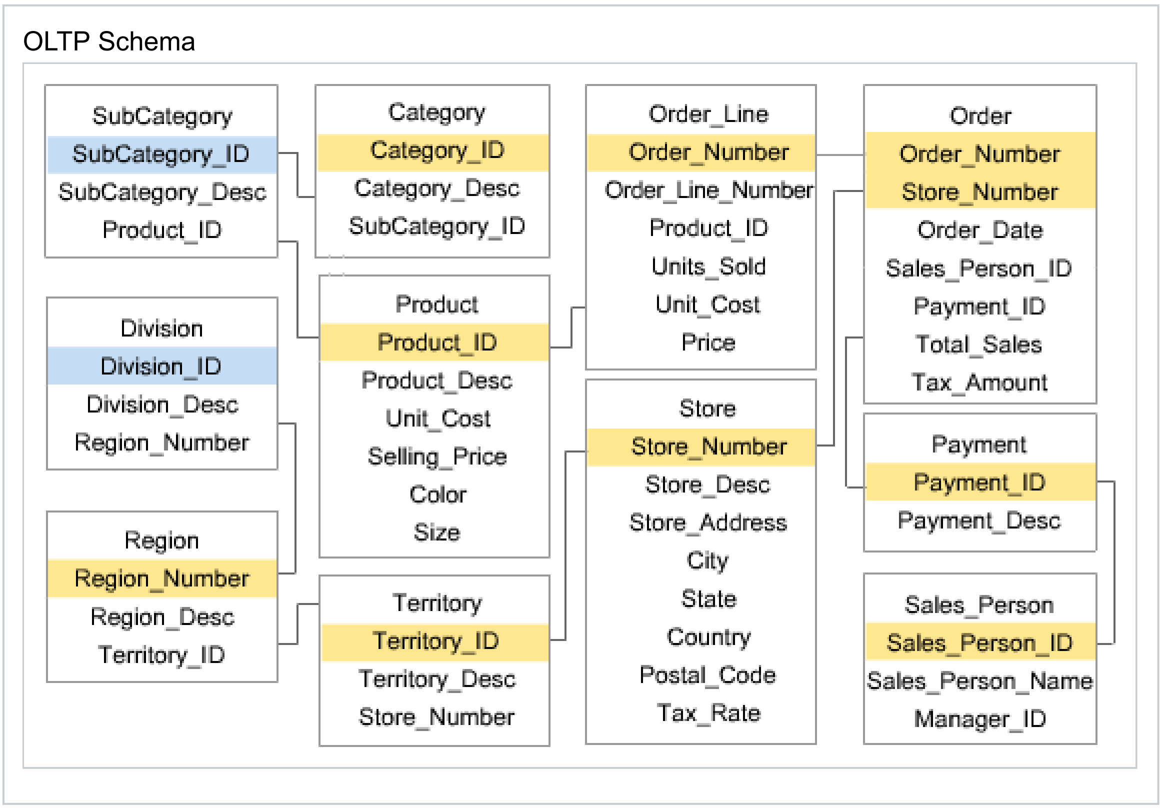
metcs779_W04_wrhs_sch video cannot be displayed here

Notice that there is one central *fact* table, and four *dimension* tables. The fact tables store data about business events, in this case sales line items. The dimension tables represent properties of those business events, in this case the time of the event, the location of the event, the customer, and the product. The fact table will have one row for each sales line item. If this is a table for the data warehouses of companies such as Sears Roebuck or McDonalds there are many billions of rows in these fact tables. There may be a row in the time dimension for each day for the last thirty years, but this is by comparison still a small table, with less than a million rows. Similarly there may be thousands of locations, which is by comparison a small number. There are also only thousands (not millions) of products offered by stores, so this dimension is also small compared to the sales line item fact. A large store may have millions of customers, but in a successful enterprise each customer has purchased many items, so this typically largest of dimensions is still small compared to the size of the fact tables.

Since typically 98% or more of the size of a star or other dimensional database is in the fact tables, there are major size, cost, and performance advantages in minimizing the size of the fact table by shortening the row length. That is part of what a good dimensional design does. The columns of the fact table that represent things such as the time of the sale, the store, the product, and the customer are represented as compact integers with foreign key references to the corresponding dimension tables. For example, instead of storing a long string such as a customer's name and address to identify the customer, we would typically store a three or four byte customer_id which references the appropriate row in the customer dimension table. Then the customer dimension table stores all of the attributes of the customer, such as their full name, address, phone number, credit card number, purchasing history, and demographics. Note that this is traditional normalization of the customer data from the fact table.

The fact table itself is not normalized in the way that an OLTP database is normalized. The fact table usually does have a primary key, which consists of the collection of dimension columns, which are colored in the figure above, so the fact table is usually in first normal form. But the non-key columns, which are uncolored in the figure above, are not normalized. There may be dependencies between the non-key columns, some of the non-key columns may depend on only part of the primary key, and in general there is no attempt to normalize the non-key columns. Dimensional schemas are optimized for query performance, and not for transaction performance. Dimensional schemas are not designed to enforce integrity constraints, so they are not used for OLTP operations.

The dimension tables in dimensional schemas are not normalized either. Dimensional tables typically include redundant information such as city, state, and postal code or date, day of week, month, quarter, and year. The attributes of the dimensions are designed to have the data that is needed to quickly identify the rows in the dimensions that are used in queries. The extra columns in the dimension tables does not have a significant effect on the size of the database, because the dimension tables are much smaller than the facts, and the extra columns in the dimension tables can substantially speed queries. For example, suppose that you are doing a marketing query against a 2 terabyte warehouse with 60 million customers, and you want to check how your sales on Tuesdays to male customers in New York compare to compare sales on Wednesdays, then it would speed your queries to have a day_of_week column in your time dimension, a state_code column in your location dimension, and a gender_code in your customer dimension.



Here is the same decision support query as earlier, against the dimensional schema:

Decision Support Query against the Dimensional Schema

```
SELECT pd.Category_Desc, td.year, td.month, sum(sf.Total_Sale) TotSales

FROM Sales_Fact sf, Product_Dim pd, Time_Dim td

WHERE pd.Category_Desc = 'beverage'

AND td.month = 12
AND td.year IN(2004,2005)

AND sf.Product_ID = pd.Product_ID

AND sf.Time_ID = td.Time_ID

GROUP BY pd.Category_Desc, td.year, td.month
```

Note that the query is much simpler, and that there are no function calls in the WHERE clauses. This query should complete within a few seconds against a well-implemented dimensional schema, even if the database is many terabytes in size.

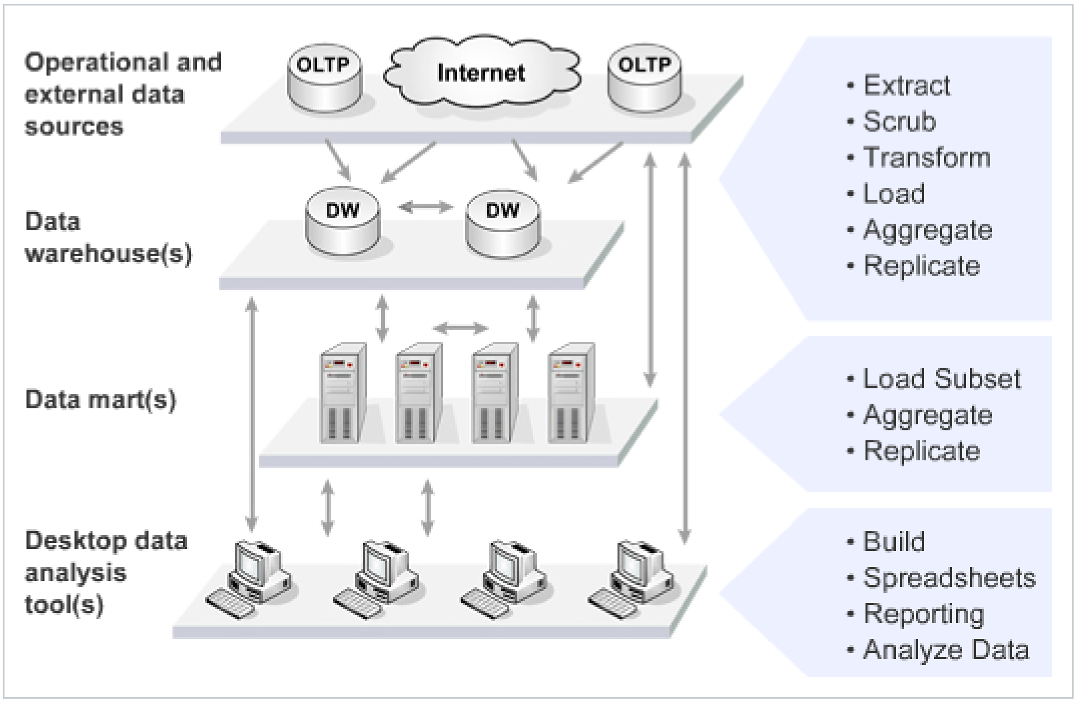
Performance Tip

The IN clause looks like a function call, but IN is part of SQL, so the optimizer understands it and doesn't need to treat it as a function call.

Data Warehouses

Data warehouses are part of a larger decision support infrastructure which is illustrated in the following figure.

Data Warehouse Components

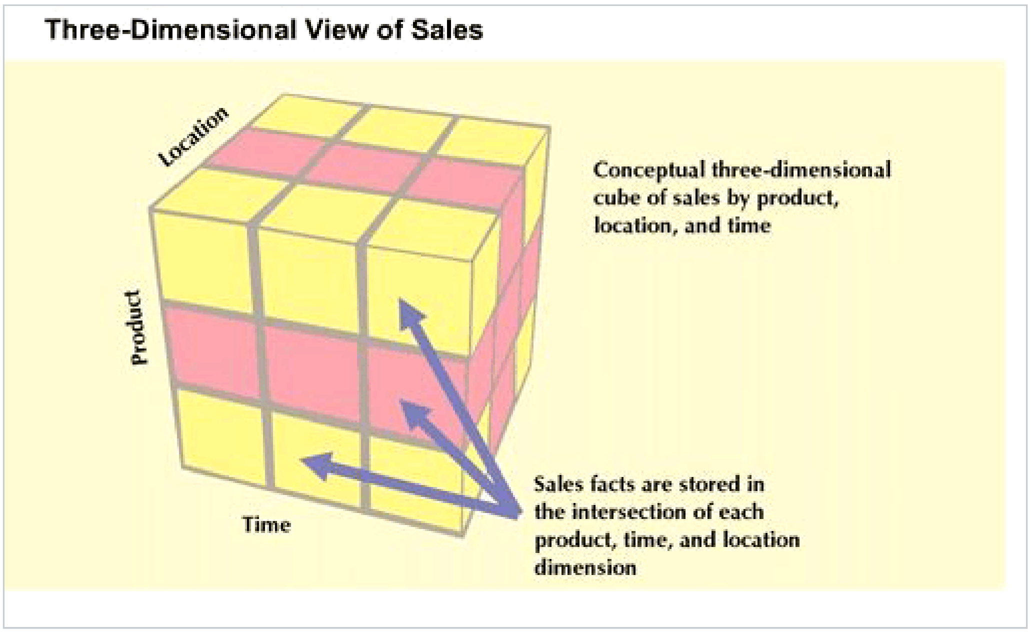


From the figure you can see that data is loaded into data warehouses from operational (OLTP) databases and from other sources, which are often on the web. This processing to load the warehouses is usually referred to as Extract, Transform and Load, or ETL. ETL processing usually includes extracting the data from operational and other data sources, scrubbing to remove errors in the data, the computation and updating of aggregates such as daily and monthly summaries, transformation of the data into a dimensional or multidimensional form, and updating of the data warehouse database.

The figure also shows how data marts are loaded from data warehouses. The data marts are often designed to support particular functional areas such as finance, marketing, production, or sales. One advantage of loading data marts from data warehouses are that this localizes in the warehouse ETL the often complex processing to extract, scrub and reorganize the data for decision support. The data is often loaded into the marts after the warehouse ETL cycle has finished for a day, using queries that fetch just the data needed by each mart. Data may also be loaded from the warehouse into the marts on demand in response to user requests. In this way the users of the marts have nearly interactive access to all of the data in the warehouses, without the costs of storing that data in the mart databases. When marts are fed from warehouses the marts do not each have to deal with the considerable complexity of extracting the data from diverse sources, scrubbing it, and transforming it into useful information.

Relational and Multidimensional Data Warehouses

Data warehouses and data marts are implemented using both relational and multidimensional database technologies. We have discussed above the relational dimensional technology. There is also another set of related technologies which are termed *multidimensional*. Multidimensional databases are based on different ways of storing the information in a data cube. The following figure from the text illustrates a three-dimensional data cube for sales data. Most data cubes used in data analysis have more than three dimensions.

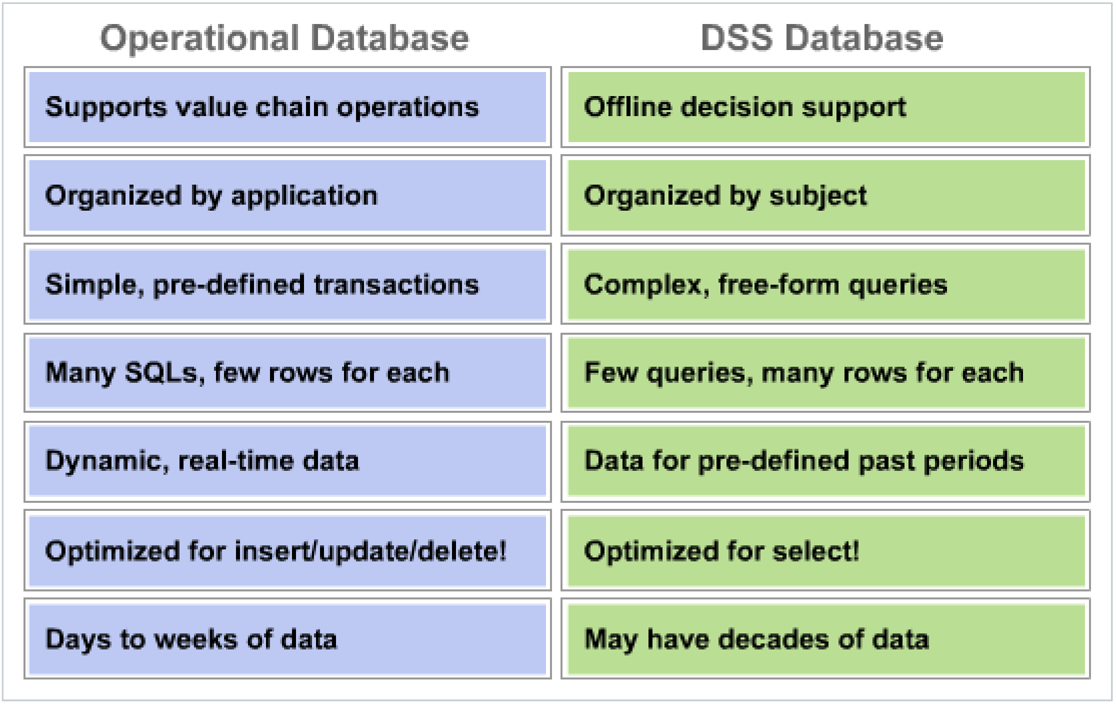

Three-Dimensional View of Sales
Location
Product
Time
Conceptual three-dimensional cube of sales by product, location, and time
Sales facts are stored in the intersection of each product, time, and location dimension

Presumably most products are not sold at most locations in most times, so that the data in the data cube is sparse; as the number of dimensions increases the sparsity increases. Early multidimensional databases used memory-resident sparse array techniques to store this data in main memory. This makes multidimensional databases very fast, because there is no I/O. It has also limited the size of multidimensional databases, and increased their RAM requirements. The following table compares relational and multidimensional databases.

Characteristic	Relational	Multidimensional
Storage	Tables in a dimensional schema	Data cubes

Plasticity	Dimensions and facts can be added at runtime	Changes require rebuilding the cube
Size scalability	Up to hundreds of terabytes	Small to medium only
Standards	Portability; ANSI/ISO standards	Proprietary
Ad hoc	Supports ad hoc queries with any number of dimensions	Predefined dimensions only
Resources	Runs well on symmetric multiprocessors	Very high RAM requirements
Performance	Good performance for all DB sizes, if well indexed	Very fast for small databases that fit in RAM

The Common Criteria OLTP Database vs. Data Warehouse



Operational Database	DSS Database
Supports value chain operations	Offline decision support
Organized by application	Organized by subject
Simple, pre-defined transactions	Complex, free-form queries
Many SQLs, few rows for each	Few queries, many rows for each
Dynamic, real-time data	Data for pre-defined past periods
Optimized for insert/update/delete!	Optimized for select!
Days to weeks of data	May have decades of data

How to Build a Data Warehouse

Something to Keep in Mind

Most enterprises fail in their first attempt to build a data warehouse. If you are planning to build your first data warehouse follow the general guidance provided here, not the guidance in the text, and hire a consultant who is skilled in warehouse development. There are many technical and political challenges involved in building a data warehouse.

The basic steps for building a warehouse are to:

1. Build the data warehouse as a sequence of snap-in data marts
2. Learn the Business
3. Design the Data Warehouse
4. Design and Implement ETL Processes
5. Choose User Tools for the marts
6. Prototype using subsets of the data

Snap-in Data Marts

The best way to build a data warehouse is as a collection of “snap-in” data marts, rather than trying to build the data warehouse in one big project. This approach lets us start with the highest return on investment (ROI) functions. Do the ROI analysis and identify the best place to start.

The Process of Developing Snap-in Data Marts	
Define the Common Schema	The process of developing snap-in data marts consists of defining common dimension tables and common time and other grains, and then defining the fact tables needed for the different marts. Then the process of adding a new mart only involves adding new columns to existing fact tables and/or adding new fact tables.
Identify the Marts	At this time it is good to identify the different snap-in marts that you plan to build, and to identify the facts and potentially additional dimensions or columns that those marts would require. This gives you a general design for the integrated data warehouse schema. You don't need to worry too much about the details of the fact tables, but you do need to identify their primary keys (to help identify duplicate fact tables). While you are doing this be careful to check that the time grain that you have chosen works for all of the potential marts. If the time grain that you choose for the data warehouse is smaller than some data marts require that will only cost a little processing. If some data marts later require a finer time grain than you chose for the data warehouse then to integrate the new marts you will need to reload all of the data in the data warehouse. Reloading a data warehouse from the raw data can take weeks, so it is best to avoid this.

Learn the Business

Interview management and end users to determine what the real decision support requirements are. Be aware that sometimes the most important decision support needs or opportunities are not known to management. Choose the business processes (data marts). Start with the highest return on investment marts; when these marts are proven successful, you can then expand the warehouse to include other functional areas and marts. There is significant advantage in completing the warehouse to support cross-functional business intelligence, so try to keep working on the marts until the data required to make enterprise-level decisions is present in the emerging data warehouse.

The Power of Data Warehouses

One of my clients was sure that they had no duplicates in their customer table, because they paid a firm each year to detect and remove any duplicates. The warehouse provided them with powerful means to identify duplicates, and they discovered that about twenty million of their sixty million customers were actually duplicates.

You should be aware of the business facts and dimensions as you learn the business, and should catalog and define them as you encounter them. Interview database and data administrators and learn the facts and dimensions that they use and all of the other things that they know about their data. Pay close attention to any known data problems, because you will need to cleanse the data of most problems before it enters the warehouse. Expect many more data problems than operational database people are aware of.

You need to pay close attention to what users expect and need from the marts and warehouse. You also need to carefully set end-users' and management's expectations about what the marts and warehouse will accomplish, and how much effort it will require.

Design the Data Warehouse

Now that we have learned the business, and identified the business facts and dimensions it is time to design, the data warehouse. From the business facts identify the facts that will go in the data warehouse. Determine the granularity of the business facts.

From the business dimensions identify the dimensions that will go in the warehouse. Identify the dimension attributes and their descriptions. Identify the aggregations and any aggregation hierarchies such as a time hierarchy with days, months, and year aggregation scales.

On Weekly Time Grain

A one-week time grain is problematic in a time aggregation hierarchy, because weeks do not roll up cleanly into months. Hours roll up cleanly to days, days to months, and months to quarters and years. The scale ratio of about thirty between days and months is not usually a problem. Most operations on weeks are better met by a day of week column in a time dimension with a granularity of one day.

Choosing a time grain

Most businesses can function with a time grain of one day, but for businesses with important variations over time scales smaller than one day a finer time grain is required. For example, the warehouses for large restaurant chains such as MacDonald's may use a time grain as small as fifteen minutes, so that they can use the warehouse to accurately forecast the demand for their products through the day.

Determine the Historical Duration of the Data Warehouse

The historical duration of a data warehouse is usually as long as there is data of useful quality available in computer-readable form. Sometimes it is worthwhile to add older paper records to the data warehouse by using optical character recognition and manual key-in.

Determine the Extract and Load Schedules

In earlier times the ETL was normally performed in the middle of the night, usually shortly after the OLTP systems had completed the last transactions for the previous day. With increasing competition and refinement of DSS systems the schedules are now often shorter, and may be as short as an hour or even nearly continuously, to capture the latest business trends and to support business processes such as real time supply chain management.

As you are designing the schema you should be aware that industries have common facts and dimensions, so that your schema may be very similar to the schema of another enterprise in the same business area. Individual enterprises may have additional facts and dimensions that are not common within the industry, reflecting differentiating aspects of their business model. Individual users or uses of a mart or warehouse may require additional facts or dimensions, and it is common to add attributes to fact or dimension tables during the operational phase of a warehouse.

Example Star Schemas

In this section we illustrate the most basic fact and dimension tables found in warehouses and marts for some common business areas. Particular businesses in these business areas will usually have additional facts, dimensions and attributes.

metcs779_W04_groc_ex video cannot be displayed here

Most of a typical retail dimensional schema is fairly obvious, but the promotion dimension may not be. Promotions of various kinds play a key role in the intensively competitive retail arenas. A promotion can increase the profits of a store not only by increasing the sales of the items that are being promoted, but also by attracting customers who buy other goods besides those which are being promoted. A promotion can also reduce profits, when a product on sale cannibalizes the sale of another more profitable product. Retail warehouses with a promotion dimension help managers determine how to best promote their businesses.

metcs779_W04_ins_ex video cannot be displayed here

This schema for insurance businesses represents the basic business facts of operations on policies, with the ubiquitous time dimension, and dimensions for the policy and the item at risk such as a vehicle or life. The Name dimension can refer to the name of an individual such as someone covered by health insurance, or the name of a corporation or other legal entity.

Design the ETL Processing

During the interviews you have been identifying the data sources and any problems with the data that need to be cleansed. It is now time to design the processing that will extract that data from each of the sources, transform and scrub it, and load it into the mart or warehouse. Often 80 percent of all data warehouse effort occurs during this step of designing and debugging the ETL processing.

It is important to realize that OLTP data must be cleansed before it is useful for decision support. In decision support we are often looking for ways to improve performance by a fraction of a percent, so small errors in the data can throw off the analysis and result in incorrect conclusions. For example, a source system may be missing a few percent of the data for a day; this can invalidate the results for any time periods that include that day, and any results that include that day in a baseline analysis.

Plan the ETL processing carefully, taking into account the times when the data sources are available, the time required to extract the data, to transform it, and to load it. Consider carefully the relationships between the data mart and enterprise models. Develop a data dictionary that defines the identifiers for all entities, including facts, dimensions, and attributes in particular, and use those identifiers everywhere. Pay particular attention to cross-business boundaries, and the differences in the data formats, precision, and quality. In the mart or warehouse you will need to reconcile all of these differences. Investigate ETL tools such as Informatica, Platinum, Carleton, Prism, and Praxis.

Finally, do a pilot with a representative subset of the data. Don't expect the pilot to work. There will be problems extracting the data. There will be problems with the data itself. When you have fixed the problems that you encountered with the small pilot, then try successively larger pilots until you have solved all of the problems and can ETL all of the data.

Select the End-user Tools for the Functional Area Marts

There are many sophisticated end user tools for decision support for business functions such as finance, marketing, resource planning, sales, and production management. It is almost never cost effective to build this functionality, so it is usually best purchased in a tool. These tools have specific data requirements which must be considered in the schema and ETL designs. Therefore some of this end-user tool selection must be factored into the design of the schema.

The development proceeds in phases, with the schema design and tool selections threading through the phases. The first phase is discovery, when you understand the business and document the facts, dimensions, and attributes. The data dictionary is one product of this phase. This phase usually produces a rough warehouse schema. The second phase is design, when the schema is defined to the create table level. During this phase the tools are also evaluated, and their fit to the schema, and the schema's fit to them, evaluated. The third phase is ETL design and piloting. During this phase more will be learned about the tools, and schema changes will occur. In the fourth phase the mart and emerging warehouse are loaded and the users begin to look at the data. In the fifth phase the mart or warehouse is used to verify the correctness of the data. This QA process continues through the life cycle of the DSS system, where we are always checking that anomalies are not due to incompletely scrubbed data.

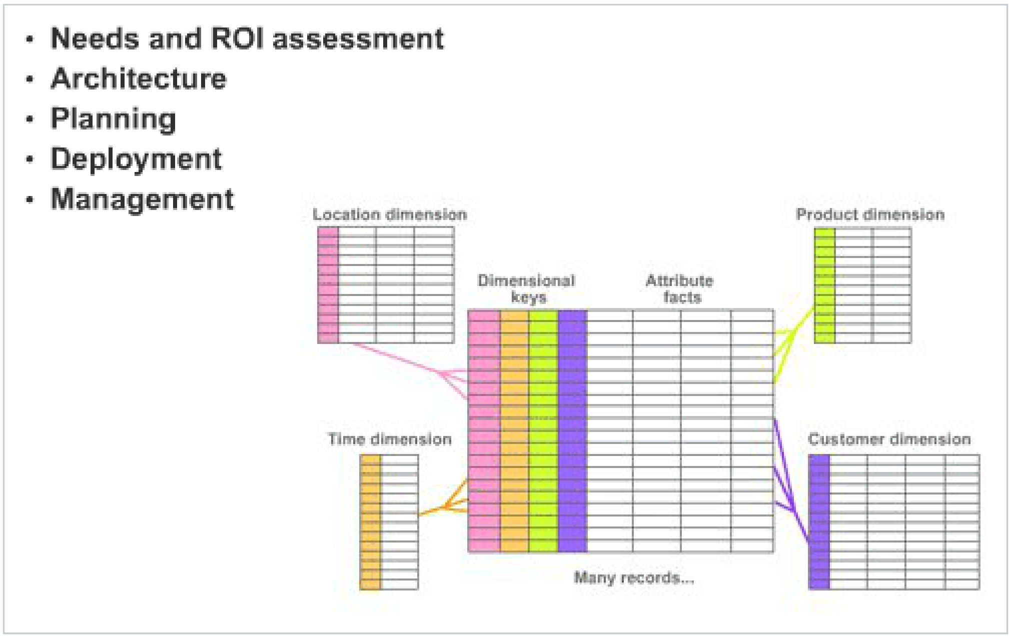
Prototype

It is important to try out your understanding and test the reality of the data by prototyping the data warehouse on subsets of the data. This is particularly important for the ETL. Prototyping gives you the opportunity to test the end-user tools on small subsets of the data early in the development, before the often lengthy process of loading all the data. Prototyping reduces risk by identifying problems as early as possible. Expect problems, and plan and budget for them, particularly in data cleansing.

Summary of Data Warehouse Development

The final figure summarizes the keys to successful Warehouse Development. Again, if this is your first warehouse, be sure to get help from someone who has been through this before.

Keys to Successful Warehouse Development



- Needs and ROI assessment
- Architecture
- Planning
- Deployment
- Management

Data Virtualization

A significant approach to gathering data is rapidly emerging. The concept of Data Virtualization refers to the use of a product or set of products to present data from disparate sources as a unified collection. This basic idea is not especially new or fresh; in loose terms, a data warehouse or mart could be considered a virtualized solution. But the positioning of dedicated tools by major vendors has given Data Virtualization a foothold in the information management landscape, and its promise is a threat to traditional ETL workflows and data warehouse approaches.

The Problem

In large measure, the business value of data analytics has supplanted the technical value. This shift has coincided with an enormous increase in data variety, volume and velocity, and this growth will continue to skyrocket as we begin to embrace the "Internet of Things". Another significant evolution has seen business intelligence usage spread far beyond the highly trained and specialized "power users" of the past. Instead, business users of all backgrounds, roles and skill sets have come to leverage and expect business intelligence as a critical success factor. In the recent past, this created a problematic gap between the rapidly changing needs of business users and the ability for technical staff to prepare customized data views and reports. To help bridge the gap, a proliferation of user-friendly BI tool suites emerged in the 2000s that finally allowed business users with limited technical or data skills to create self-service queries and reports. But despite this step forward, a crucial problem remained unresolved: data integration. The best and most efficient data warehouses still require significant effort, expertise and expense to efficiently pull data from disparate sources together. ETL processing, which is most often needed in data warehouse workflows of even moderate complexity, is notoriously costly and difficult. In many ways, the end users of the business data are still beholden to the capability, expertise and capacity of technical staff just as they were when custom reports had to be generated for them. It is in this environment that Data Virtualization has emerged as a powerful solution to empower users while reducing cost and complexity.

The Promise of Data Virtualization

Data Virtualization tools are software products that run on dedicated servers. This technology presents enormous potential by virtualizing both sources of data and the types of sources of data.

Data Source Virtualization: Data Virtualization tools to gather all disparate data and present them in "virtualized" form, giving the end user the impression that all of the data came from the same source. The underlying configuration of these tools is typically the responsibility of BI data experts, as there is a great deal of technical customization and tuning that can be done by those with advanced knowledge of the product. Data owners and custodians can use the Data Virtualization product to assemble, curate and package federated data views, though the real power of Data Virtualization is that this responsibility is optional. The end users of the data can choose to leverage curated data assemblies or create their own views based on their individual needs and preferences. This ability to create data "mashups" is an empowering, revolutionary advance that fulfills the promise of shifting the value proposition of business intelligence from the technical side to the business side.

Data Source Type Virtualization: Another challenge in today's data economy is that data is much more than records in relational databases. The "data lake" from which business users wish to transform data into information is vast, and it includes not just RDBMS but NoSQL, Big Data clusters, APIs, social media streams and more. It usually contains a mixture of structured, semi-structured and unstructured data. A traditional data warehouse is not well equipped to handle this diversity, but the prominent Data Virtualization tools on the market can handle these various data sources with relative ease.

How Data Virtualization Works

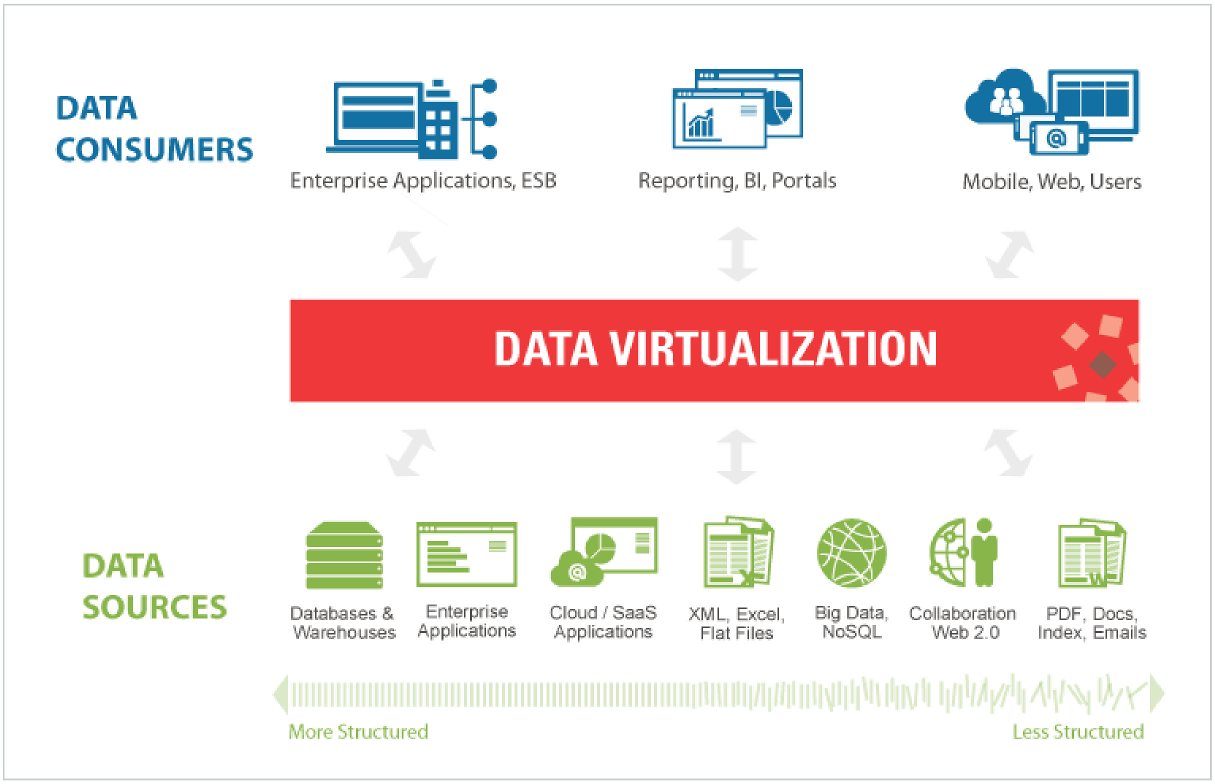
Data Virtualization tools use standards and widely accepted protocols. For example, an organization may choose to expose its federated data via ODBC, JDBC, OData, web services, RSS and more. This makes integration with popular BI tools easy and removes some of the obstacles end users often face when attempting to hook their favorite BI and data visualization tools up to external data sources.

Major vendors such as Cisco, IBM and Informatica have significant investments in Data Virtualization tools, and the technology is also the focus of Data Virtualization "pure play" startups such as Denodo Technologies.

Data Virtualization products need not be considered as replacements for entrenched and functional data warehouses, applications or tools. They can bring together these existing systems without affecting the source systems in any way. But many organizations will find that Data Virtualization may render legacy systems and processes obsolete, with enormous cost-saving potential. For example, a large organization that builds and maintains many ETL jobs to move data from source systems into warehouses or other databases may discover that by pointing their Data Virtualization software directly at the source systems, the need for ETL processing is obviated because loading is no longer necessary. Extraction and transformation can be handled by the Data Virtualization tool itself, or by end users who wish to customize the transformation aspect of the target data.

The following diagram from Denodo Technologies shows the role a Data Virtualization product can fill:

Data Virtualization



© Denodo Technologies, http://www.denodo.com/sites/default/files/public/images/dv_architecture_design-time_simplified_0.png

Extra Topic - Web Technology and DBMSs

Database Connectivity and Internet Databases

metcs779_wk6 video cannot be displayed here

Learning Outcomes

By reading this lesson, participating in discussions, and completing the assignments, you will be able to:

- Explain e-commerce and the effects of e-commerce on business
- Describe electronic commerce styles such as B2B and B2C
- Describe the architectural components required to conduct electronic commerce over the internet
- Describe database design and implementation issues for e-commerce applications
- Describe, read and produce Extensible Markup Language (XML) documents
- Describe the many roles of XML in modern e-business
- Understand different ways to create XML documents
- Understand how to use XPATH and FLWOR to process XML documents in databases

Introduction to Internet Commerce

This material is provided for students who may not understand the technologies underlying internet business and internet databases.

What is internet commerce or e-commerce?

Electronic commerce (also called *e-commerce* or *internet commerce*) is the use of electronic networked computer technology to market products and services and to improve the efficiency of business operations, including value chain integration. Most e-commerce transactions take place over the internet and between businesses, where they are used to automate and streamline business transactions. For example, many modern businesses automate their supply chains using e-commerce technology and the Internet. The many web-based retail businesses such as Amazon and EBay are based on electronic commerce technology. E-commerce is now a required part of essentially all competitive businesses, including retail, wholesale, manufacturing, and services.

The History of E-Commerce

In the 1960s banks created private telephone networks for electronic funds transfer (EFT). In the late 1960s the ARPAnet was developed at Bolt Beranek and Newman, Inc. in Cambridge, Massachusetts under funding from the Advanced Research Projects Agency. This research produced the TCP (Transport Control Protocol) protocol suite. This was extended in a few years by the addition of the IP (Internet Protocols) protocols, which allow networks to communicate. The goal of this research was to develop a computer communications network that could survive nuclear attack. As it turned out those requirements are not far from what it takes today for a computer network to survive hackers and denial of service attacks, so it is not surprising that the TCP/IP protocols (commonly called the Internet protocols) became the de facto world standard for wide area communications.

In the 1970s banks began to provide ATM services to their customers over their networks. In the late 1970s and early 1980s Electronic Data Interchange (EDI) emerged. EDI is a communications protocol that enabled companies to exchange business documents over telephone networks. This was a comparatively expensive technology so it has been largely replaced by Electronic Funds Transfer over the internet. In the early 1980s and through the 1990s business discovered the Internet and personal computers facilitated rapid expansion of the Internet. With the development of the World Wide Web the protocols were liberated from host-based addressing and liberated from the need to have specific applications on client machines. This led to very rapid growth of the internet and web. Internet technology advancement is continuing today with the emergence of improved security, sessions for the web, universal acceptance of XML and XML-based interchange standards, electronic retail and wholesale marketplaces, portal technology, and many other technologies that improve and extend the reach of the internet, web, and e-commerce.

The Impact of Internet Commerce

The internet, web, and e-commerce have had a revolutionary effect on almost all enterprises. Most businesses that have survived the internet revolution have a web presence. The internet and web have reduced the barriers of geography and have created a highly competitive global market that has attracted and created millions of businesses and organizations.

Value Chain

The value chain of an enterprise is the sequence of steps from the inputs to the outputs that provide value to those whom the enterprise serves. For example the value chain of a grocery store extends from the ordering of the goods through to the checkout counter. The value chain does not include ancillary functions such as human resources, advertising, finance, or accounting.

E-commerce and the internet not only link businesses with each other. For example, the internet is the normal way to implement near real time value chain integration.

The Internet and web are mainly used within enterprises. For example a few years ago Ford Motor Company bought their employees personal computers so that they could access the company portal and so reduce the costs of things like mailing pay stubs, and improve employee integration into the business.

Intranets

Most internet use within enterprises is conducted over an intranet, which is an internet restricted to the enterprise. For example, as I write this I am at home, with my computer connected to the Boston University intranet using a virtual private network (VPN) connection that provides secure communication over the internet. What this means is that I have the same access to the University computer network and resources as if I were in my office.

Extranets

Extranets are intranets that allow access to other specific organizations. Extranets are frequently used in value chain integration between enterprises. For example, a manufacturer may allow their suppliers read-only access to their supplies inventory data so that the supplier can more efficiently implement just in time delivery. Similarly a manufacturer may allow their customers to read their production tracking data so that their customers can follow the progress of their orders. We will discuss the roles of the internet and databases in value chain integration in section.

Advantages, Disadvantages, and Risks

The internet and e-commerce have many advantages and a few disadvantages and risks. The Internet makes comparison shopping almost trivial, with websites that crawl for the best deals for us. This makes the internet a fiercely competitive marketplace. Because internet operations require none of the expensive real estate, buildings, or labor of traditional commerce, the costs of internet vendors are usually much lower, and prices on the internet are usually much less than from traditional suppliers. This has led to a situation where the internet vendors have become the low cost vendors in most markets where shipping is not a major cost. The head-to-head price competition has forced internet vendors to minimize all costs, so there is often essentially no support available from internet vendors. The internet marketplace is gradually maturing with different vendors providing different levels of price and support, but quality branding and market recognition are still in the early stages of development.

The convenience and nonstop operation of most websites provide a significant advantage for most consumers. You can order from a web-based company from anyplace that has internet access, including aboard airliners, ships, and in many restaurants. This global access to a site, and the access of businesses to customers around the world, has made the Internet the largest marketplace the world has ever known. The size of this marketplace has been matched by a wealth of services that increase customer and vendor knowledge, and raise significant new issues of privacy.

The internet and e-commerce pose significant, often hidden, risks for both the vendor and customer. One of these is hidden costs. It is not too difficult for a vendor to add up the costs of the goods, the cost of obtaining a customer, and the cost of delivering to a customer. Vendors can overlook their vulnerability to technical failures such as outages of their sites or the sites of their business partners. Vendors or their customers can be subject to attacks by hackers or infection by viruses, worms, or spyware. These can be very difficult and expensive to eradicate. Both vendor and customer must be aware of the security risks of internet business. Some of these risks are subtle, so both vendor and customer must pay a fairly high price to become sufficiently aware of the risks and how to avoid them. This makes internet-based business unsuitable for many customers. There are also genuine privacy issues in internet transactions, so both buyer and seller must be aware of these risks and how to minimize them. There are significant legal issues such as copyright and patent infringements in many internet transactions, and many of these legal issues have not been fully resolved in the courts. Finally, identity theft and fraud are growing e-commerce problems.

Internet Business Styles

Internet business transactions have been classified according to the kinds of entities that are parties to a transaction. This is summarized in the following table:

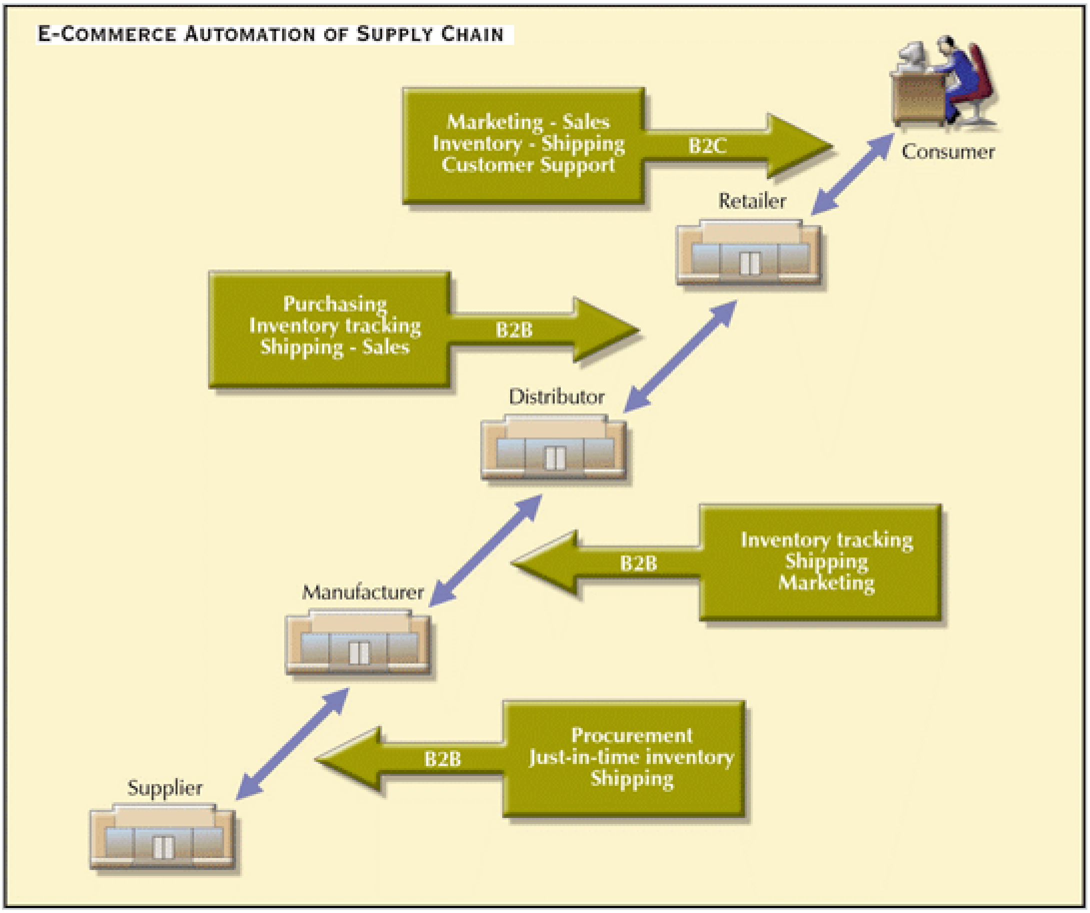
Transaction Type	Acronym	Definition
Business to business	B2B	Electronic commerce between businesses, sometimes over an extranet
Business to consumer	B2C	E-commerce between business and consumers
Consumer to business to consumer	C2B2C	Transactions between consumers mediated by a business such as eBay
Business to business to consumer	B2B2C	Business conducted through a third-party business to a consumer.

Intra-business		Internet activities within an enterprise, often over an intranet
Government to consumer	G2C	Interchanges between government organizations and people
Government to business	G2B	Interchanges between government organizations and businesses

Business to Business

B2B transactions are the most common of the Internet business transactions.

E-Commerce Automation of Supply Chain



The most successful businesses are closely coupled with their suppliers and customers using the internet. The consequences of a transaction such as a sale may ripple up the supply chain through several businesses, sometimes within a day. This close coupling reduces the capital and other costs of goods in the supply chain, the costs of unsold goods, the capital costs of goods, and the agility of the entire distributed enterprise. It also requires flexibility on the part of all partners. For example a customer may place a large order with the retailer in the figure. The retailer's computer systems place an order with their Distributor, who immediately informs the Manufacturer, who then immediately informs their suppliers of the additional materials required. The overall effects of this close integration are improving the responsiveness to consumer demand and minimizing the amount of raw material and manufactured goods in the supply chain. This allows the whole distributed enterprise to respond more quickly to changes in consumer demand and new products while minimizing the organizational lethargy and costs associated with the less responsive earlier approaches based on large inventories and paper-based order processing.

Most B2B transactions are implemented over the internet using XML and related technologies, which we will study later in this chapter. XML technologies are directly supported in modern commercial DBMS such as Oracle, DB2, and Microsoft SQL Server.

Business to Consumer

You are all familiar with B2C websites that offer products, such as Amazon.com and Dell.com. There are also many B2C websites that offer services, such as Google.com and weather.com. The latter are usually free to the consumer, with the revenue provided through advertising, just as with commercial radio and television. Many service-oriented B2C sites, such as weather.com and Morningstar.com, also offer a premium subscription service.

Consumer to Business to Consumer

There is a growing area where businesses and organizations provide services by connecting consumers or small businesses that wish to provide some product or service to consumers and small businesses that want that product or service. The pioneer and leading player in the C2B2C arena is eBay. There are many more markets in which this model has yet to expand. C2B2C has created new geographically large marketplaces in which almost anything can be traded efficiently between individuals and small businesses, with larger businesses and organizations as the low-cost intermediaries in the transactions. C2B2C has created the first reasonably efficient markets for uncommonly traded specialized items, so C2B2C has had a profound impact on tens of thousands of small markets. Today if someone wants to determine what something is worth and whether anyone wants to buy or sell something they commonly check on eBay or a more specialized C2B2C site. The economic, legal, regulatory and taxation implications of global C2B2C marketplaces are now being explored in society.

C2B2C and its commercial and often overlapping cousin B2B2B are taking on the information aspects of traditional wholesale and retail distribution organizations. There are important differences between traditional. The overheads imposed by online C2B2C and B2B2B businesses are of the order of one percent of a transaction, where the overheads of traditional organizations that make matches between buyers and sellers include the 6% transaction overhead imposed by real estate agents and the more than 30% transaction overhead imposed by most traders in smaller markets such as antiques or collectables. The C2B2C revolution has just begun.

Government, Education, and Other Organizations

Organizations of all types have a web presence. The course, which you are now taking, is one example of the extent to which the internet and web have created new opportunities and made our lives better. As you have also undoubtedly noticed, the internet not only globalizes our activities, but also lets us work, trade, or study any hour of the day or night from anywhere that has internet access.

Boston University Metropolitan College