ORACLE8*i* *INTER*MEDIA: MANAGE STRUCTURED AND UNSTRUCTURED DATA FOR FAST AND ACCURATE RETRIEVAL

By
Lin Sun

A master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in
Information Science

Chapel Hill, North Carolina

April, 2001

Approved by:

_____
Advisor

Lin Sun. Oracle8*i inter*Media: Manage Structured and Unstructured Data For Fast and Accurate Retrieval. A Master's paper for the M.S. in I.S. degree. April, 2001. 76 pages. Advisor: Gary Marchionini

This paper describes a novel design and implementation of manipulating structured and unstructured data in the Oracle8i database over the web interface using Java Server Pages. A text recognition script is developed to automate the process of piping data from text formats into the various table fields into the Oracle 8i database. Familiar Internet Explorer 5.0 or above and Netscape 4.7 or above are supported to run the web interface and perform Boolean and some other advanced search over the Oracle8i database.

A three-tier architecture is used in the web database design. An Oracle8i database stores raw data in the tablespaces. The JSP server generates the response by querying the Oracle8i database and provides the web server with data in standard HTML formats. The client side tier (web browser) inputs the query and requests the response from the web server.

Headings:

Data Piping Method – Text Recognition Script

System Structure – Three-Tier Architecture Design

Unstructured Data – Oracle8*i inter*Media

Table of Contents

Chapter 1: Introduction

1. Introduction:

Oracle8, which is a traditional database management system, provides a variety of data types you can use to create database applications that take advantage of structured data and unstructured data. Besides some general data types like date, string, number, and boolean for structured data, Oracle8 has several large object (LOB) data types like character LOB (CLOB), and binary LOB (BLOB) to support applications that must manage large unstructured objects as well as binary file (BFILE), which stores LOB locators.

Oracle8*i* is built on Oracle8, and it is known as the database for Internet computing. It changes the way that information is managed and accessed to meet the current high demand of Internet data transportation and retrieval. It provides significant new features for traditional online transaction processing and data communications between more than one databases compared with Oracle8. It provides many advanced new tools like *inter*Media, WebDB and so on to help users successfully manage all types of data stored in an Oracle database and deliver the database content, including very Large Objects (LOB), to remote users' client machines with high performance, scalability and security.

In the past ten years people have invested very heavily in building applications that enable us to rapidly retrieve structured data, which is stored in columns in the database. However, in Oracle8 a beginner's guide, it points out that many studies state that 90 percent of the world's date is unstructured. It is not surprising to find out that almost all the articles, web pages, e-mails, and other documentations are unstructured data. How wonderful if we could be able to retrieve the results of users' documentations based on users' queries!

The Oracle8*i* Server *inter*Media allows businesses to manage and access multi-media data, including image, large text, audio, video, and spatial (locator) data. The key option of *inter*Media that we will investigate in this paper is *inter*Media's text management solution that enables you to manage unstructured text information resources as quickly as you manage structured date. It allows your Oracle8*i* server to deal with unstructured data, allowing users to access the large quantity of the unstructured data. Oracle8*i* *inter*Media is revised after Oracle8 Server Context option with more powerful functions included.

Oracle Corporation announced, "Oracle8*i* is designed to access and manage all your data using the style and infrastructure of the Internet. Oracle8*i* is the most complete and comprehensive platform for building, deploying, and managing Internet and traditional

applications. Oracle8*i* provides the lowest cost platform for developing and deploying applications on the Internet."

2. Background

North Carolina's top two public research universities have launched a unique educational program to stimulate entrepreneurship by developing donated real-world technologies. This program is named the Carbon Dioxide Patent Assessment, Acquisition and Transfer Initiative (PAATI). It combines entrepreneurship, law, business, information science, chemistry, and chemical engineering expertise to commercialize technologies donated to the universities from U.S. corporations. The University of North Carolina at Chapel Hill (UNC-CH) and North Carolina State University (NCSU) launched PAATI in May of 2000 to establish a portfolio of donated CO2-related patents. The Initiative represents the first proactive effort in academia to provide technology transfer and entrepreneurial training to students by involving them in the Business to University transfer of intellectual property. PAATI brings together the engineering, chemistry, law, business, and information science expertise at both universities to identify desirable patents; to develop donation proposals, and to form commercialization plans for donated technology, which may be bundled with university patents for licensing.

Currently, the US Patent and Trademark Office (USPTO) has provided an official web database for all US patents information, which can be reached at

http://www.uspto.gov/patft/index.html. Besides, IBM has also built a nice patent searchable website (http://www.patents.ibm.com) and Cartesian Products, Inc.'s website (http://www.getthepatent.com/) can deliver the complete multi-page USPTO, EPO, and WIPO (PCT) patent documents direct to your desktop. Why do we need to build another web database if some other web databases are available to the public?

Because none of these web databases has provided a satisfactory interface or search results to professional chemists. For example, USPTO only has limited advanced search and doesn't provide search within a search, graphics or science added value. None of these features are provided by Get The Patent, which requires an initial payment. Dialog could provide search within a search, but it requires training and web access is not available.

Chapter 2: Statement of the Problem

Although the United States Patent and Trademark Office and some other companies have already built some web databases, which can provide full text search and Boolean search features, none of these systems fit the needs of those people who have specific interest in CO2 related patents. Users, especially those chemistry specialists, usually complained about the difficulty of finding the information they need. They would like to have a web-accessible carbon dioxide related database, which will provide the useful features like advanced search, search in a search, science value-added, business intelligence and be free of charge. It should be very reliable, user friendly, support graphics and make it easy to capture patents. For example, they would like to know who is the leader in the CO2 field, what countries are very active in the carbon dioxide field, patent expiration/donation information, and so on. They would also like to search on patents' full text files as well as patents' specific information like claims. Furthermore, since users have chemistry knowledge rather than information retrieval skill, they would like to have a friendly web interface that makes it easy to learn how to search the database remotely.

Another problem associated with the carbon dioxide related patent information management system is that there exists more than 1600 patents now, and the number of patents are growing very year. The patents are in text format when downloaded from the Dialog database, which means that neither can users search on a specific field like claim

nor can users search on the full text before the text format data is piped into the database in different data types. Furthermore, the database back-end should be very easy to extend or interoperate with other database management systems.

Moreover, there are a couple of performance issued involved with the carbon dioxide related patent information management system. For example, besides a friendly user interface, users request fast and accurate information retrieval and group users' customized interface if possible and three level of security (view only, write/update, and administrate) to the database. The database interface should be password protected with username and password determining security level and privilege. In the meanwhile, users also request retrieval words based on their occurrences ranking in the whole database or specific fields like claim.

All in all, how to provide fast, accurate and reliable retrieval for the structured and unstructured data in the database management system becomes a very important research question in the PAATI program. The Oracle8*i* Standard Edition on a Unix Platform has been proposed as the database backend for its wide ranges of data types support and its ability to access and manage all data using the style and infrastructure of the Internet. Java Server Page, which is a part of the Java™ family and enables rapid development of web-based applications, has been proposed as the web interface language for its platform independence. After the JSP and Oracle8*i* server marry through Oracle JDBC (Java Database Connectivity), a dynamic web database will be generated to allow users to view, query and update the database.

Chapter 3. Literature review

1. Information Retrieval Overview

Information Retrieval is a very wide term, and I am only concerned with automatic data & document in the Oracle8*i* database retrieval system in this paper. Mr. David Blair (1984) has mentioned, “The computerized retrieval of documents or texts from large databases is an area of increasing concern for those who design or use information management systems.” The dramatic growth of emails and web documents will require supplicated information retrieval system, if users would like to query these documents.

The design and implementation of large unstructured document retrieval have lagged behind those of small structured data retrieval. Blair also pointed out that people usually treat the logic foundation and technology of large document retrieval and the structured data retrieval system the same. However, he thinks they are significantly different “in how the queries are answered, in the relationship between the formal system request and user satisfaction, in the criterion for successful retrieval, and in the factors that influence retrieval speed”.

Basically, queries to the structured data are direct, the responses are relatively fast and the retrieval speeds are contingent on the physical searching speeds of the system. Also, the

structured data retrieval will provide more response on relative or irrelative data, which means it will usually retrieve only the relevant answers and the criteria of success is correctness. However, there is not a distinction between correct and incorrect for the large document retrieval. Large documents are generally retrieved with their relative scorings. The queries are indirect and the retrieval speed is more dependent on the number of logic decisions the user makes in the search.

2. Backend--Oracle8*i* Database

Compared with all the database management systems available in the market, we think Oracle database has great advantages over other database systems in reliability, platform independence, and compatibility with most programming languages. Oracle8*i* is the newest production product of Oracle Corporation, and it is available as Oracle8*i* Standard Edition, Oracle8*i* Enterprise Edition, and Oracle8*i* Personal Edition. Because the School of Information and Library Science could obtain university license from Oracle Corporation, we will use the Oracle8*i* standard edition for our projects. Currently, Oracle8*i* is running on a Solaris UNIX box called Topaz and *inter*Media is one of its key options.

Oracle8*i* *inter*Media content is stored in tablespaces on the Oracle Server. It could be image, audio, video or documents and can be within the database, in flat files or behind a web URL, but always catalogued by Oracle8*i* *inter*Media. Compared with Oracle8's ConText, Oracle8*i* *inter*Media's text services are more tightly integrated with Oracle8*i*.

There are no servers to start up, there is no query rewrite, and index creation is done through familiar SQL rather than through a custom PL/SQL interface.

Oracle8*i* *inter*Media supports a wide range of data types, but in this paper we will focus on the varchar2 (4000) and CLOB data types. After using Oracle8*i* *inter*Media to index the different data types, we could provide diverse functions to the users using Oracle8*i*'s CONTAINS query, which can only appear in the where clause of a select statement and never appears in the where clauses of insert, update, or delete. The CONTAINS function provides the following features:

- Exact matches of a word or phrase
- Exact matches of multiple words, using Boolean logic to combine searches
- Search based on how close words are to each other.
- "Fuzzy" matches of words

3. Front end--Java Server Pages

With the appearance of web database technologies—Common Gateway Interface, Active Server Pages, Cold Fusion, Personal Home Pages and Java Server pages, web pages are no longer static, but could be dynamic with communication to a back-end database server. Currently, websites are able to display and manipulate the information lying on the database server.

Among these five web database technologies, Java Server Pages is the newest technology. JSP, which is an extension of the Java™ Servlet technology, is praised by Sun

Microsystems as “platform independence, enhanced performance, separation of logic from display, ease of administration, extensibility into the enterprise and most importantly, ease of use.”(http://java.sun.com/products/jsp/index.html, 2001). It allows web programmers/designers to easily develop and maintain the information-rich, dynamic webpages. It also separates the user interface from content generation, which enables designers to change the overall page layout without altering the underlying dynamic content and informing the web programmers.

Java Server Pages (JSP) is very similar to Active Server Pages, but it is written in the Java programming language and inherits many advantages of Java™ like encapsulation, platform independence and hiding information. Most importantly, JSP logic could reside in server-based reusable resources like JavaBeans™ . Sun Microsystems says, “By separating the page logic from its design and display and supporting a reusable component-based design, JSP technology makes it faster and easier than ever to build web-based applications.” (http://java.sun.com/products/jsp/index.html, 2001).

Another great advantage of JSP is that Sun Microsystems has made the JSP specification freely available to the development community. JSP pages share the "Write Once, Run Anywhere™" characteristics of Java technology. Tomcat, which is integrated with the Apache web server, is the JSP and Servlet Engine. Tomcat 3.2.1, is the latest release quality build and it is available at http://jakarta.apache.org/ at no charge.

Chapter 4: System Analysis and Design

1. System Analysis

In order to build a CO2-related patent information management system, we need to build a Carbon Dioxide related patent management information system based on users' searching behavior and information needs. We will provide many new features like advanced search, search in a search, science added value, and most importantly, easy of use. Our user will be chemists, related universities, companies and research institutions. We are centering on the following aspects of our system:

- Database backend--reliability and extensibility
- Fast and accurate information retrieval.
- Platform independence.
- Friendly and easy to use web interface.

Currently all the patents are in text formats. Basically, they are all following the same template to structure their data. For example, they follow the exact sequence of title, patent number, inventors, and so on. Some of the patents have post-issued assignees, some of them have priority—foreign information, some don't. It is tedious and painful for humans to type in all the data into any relational database management system.

First of all, we analyzed the template for every patent, and got the following templates. But as we noted before, not every patent follows exactly the same template, which is very reasonable to us because not all patents have foreign information or continuation information. Almost all the values are required, except that Post-issuance assignments, priority, and patent continuation information are the optional values, which give us some challenges in automating the data piping process. Further, clients desire to have detailed information like inventor's first name and last name instead of block information, and this gave us more challenges in the automate piping process.

Utility
[Title]

PATENT NO.: [patentNum]
ISSUED: [issuedDate]
INVENTOR(s): [list of inventor(s), including name, place and country]
ASSIGNEE(s): [list of assignee(s), including name, place and country]
EXTRA INFO: [extra information]

POST-ISSUANCE ASSIGNMENTS

ASSIGNEE(s): [list of assignee(s)]

APPL. NO.: [application number]
FILED: [application filed date]
PRIORITY: [foreign information, include foreign patent num, country and issued date]

[patent continuation information]
FULL TEXT: [line of text]

ABSTRACT

[abstract]

What we claim is:

[list of claims]

Second, we designed a database schema based on patents' template and values. Normalization is the key design consideration when we are doing the schema design, because the database is going to grow every year as new patents come in. Reliability is also a consideration, and we believe that the Oracle8*i* database server on a UNIX platform will provide us much more reliability than other database servers.

2. Database Design

a) Database Schema Design

Because Patent number is unique to every patent, we decided to use it as the primary key for every patent. Since assignees and inventors could appear in many patents, I decided to use individual tables to store assignee and inventor information to make it re-usable. After I gave careful consideration to optional values, multiple values and values relationships, I designed nine tables to store the information in every patent. Please note that the primary key is underlined in every table, and foreign keys are pointing to their primary keys.

It is not difficult to find out that this database schema design is well normalized, and easy to extend if we want to add more text descriptions or image files to every patent. We could create a new table and use patent number and other information as the primary key, and refer the patent number in the new table to the patent number in the patent main table.

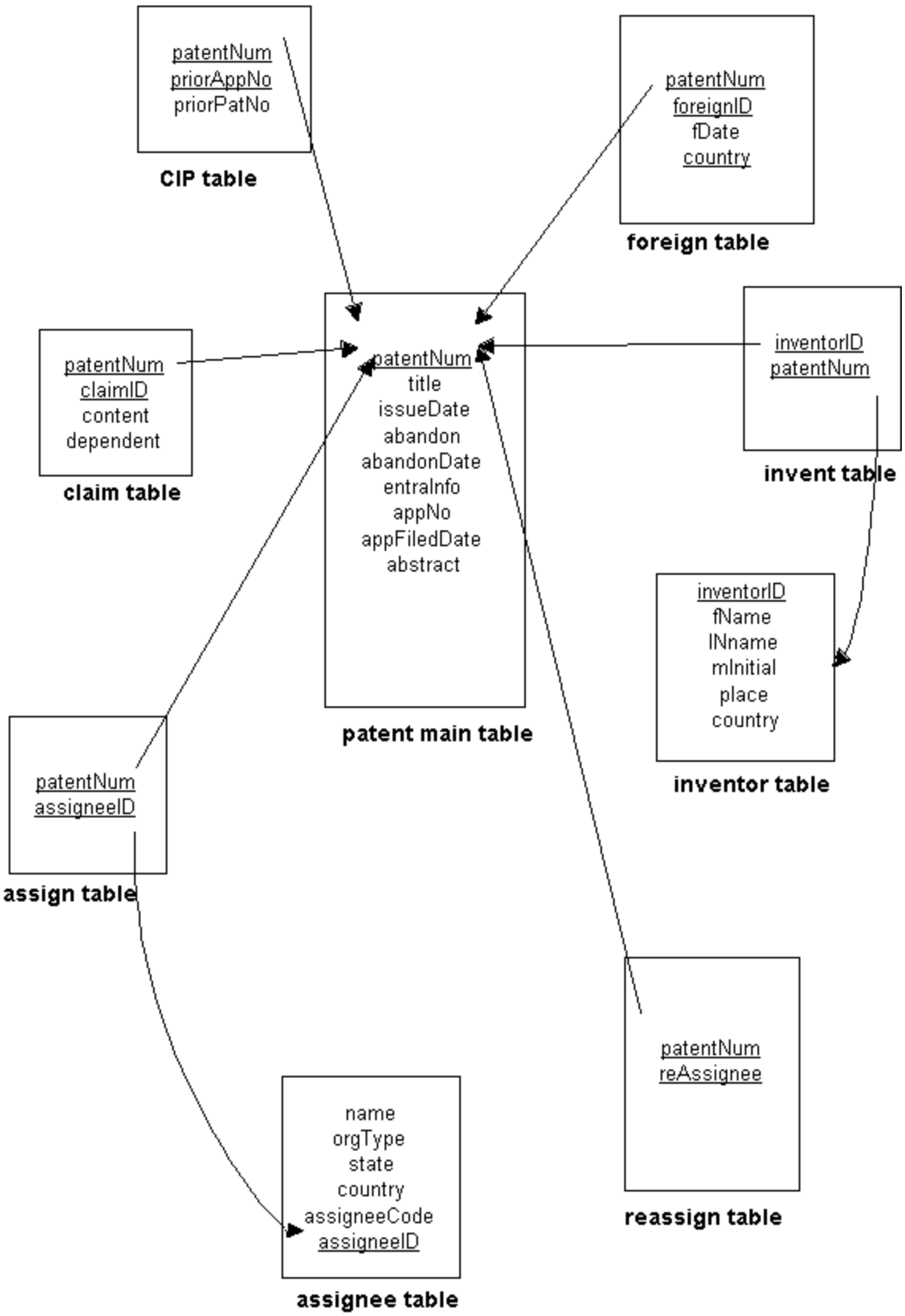


Fig 1. Patent Database Schema Design

b) Data Dictionary

For a patent, we need to keep track of the title, patent number, issued date of the patent number, the inventor(s) (name, place and country), and assignee of the patent. Besides that, there may be some extra information for a patent, like the expiration information, or if the patent has been reassigned. Furthermore, a patent always has an application number, which might be a continuation of another application number, and we also need to keep track of that and the application filed date too. Sometimes a patent has a foreign Number, so we need to keep track of the country and the patent number in that country. At the end, we need to keep track of the abstract and claim of the patent.

The CO2 related patent database keeps track of all the information mentioned above. It has nine tables: **patent, invent, inventor, CIP, foreign, claim, assign**, **assignee** and **reassign** table. The main table is the patent table.

- **Patent table:**

The patent table is the main table of the patent database. The patent table has nine attributes: patentNum, title, issueDate, abandon, abandonDate, extraInfo, appNo, appFiledDate and abstract. The primary key is patentNum.

Field Name	Description	Type	Primary Key	Foreign key	Required?
patentNum	Patent number	NUMBER	Yes	No	Yes
Title	Title of the patent	VARCHAR (200)	No	No	Yes
IssueDate	The patent issued date	DATE	No	No	Yes
Abandon	If the patent is	CHAR (1)	No	No	No

	expired				
abandonDate	The expired date if the patent is expired	DATE	No	No	No
extraInfo	Extra information of the patent	VARCHAR (4000)	No	No	No
appNo	Application number of the patent	NUMBER	No	No	Yes
appFiledDate	The application filed date	DATE	No	No	Yes
abstract	The abstract of the patent	VARCHAR (4000)	No	N0	Yes

Table 1. Patent Table Design

- **Inventor Table:**

Each different inventor is given a different identified number. Some patents might have more than one inventor. Some inventors might appear in different patents. The inventor table and invent table keep track of the inventor's information and the relationship with the patents.

The inventor table has six attributes: inventorID, lName, fName, mInitial, place and country. The primary key is inventorID.

Field Name	**Description**	**Type**	**Primary Key**	**Foreign Key**	**Required?**
inventorID	Inventor's identification number	NUMBER	Yes	No	Yes
lname	Last name of the inventor	VARCHAT (25)	No	No	Yes
fname	First name of the inventor	VARCHAT (25)	No	No	Yes
mInitial	Initial of the middle name	CHAR (1)	No	No	No
place	Place where the inventor comes from	VARCHAT (25)	No	No	No

Country	Country where the inventors comes from	CHAR (15)	No	No	Yes

Table 2. Inventor Table Design

- **Invent Table:**

Only two attributes—inventorID and patentNum are included in the invent table. The primary key is the combined inventorID and patentNum.

There are two foreign keys in the invent table: the foreign key invent.inventorID references inventor.inventorID and the foreign key invent.patentNum references patent.patentNum.

Field Name	**Description**	**Type**	**Primary Key**	**Foreign Key**	**Required?**
inventorID	Inventor's identification number	NUMBER	Yes	Yes	Yes
patentNum	Patent number	NUMBER	Yes	Yes	Yes

Table 3. Invent Table Design

- **CIP Table**

CIP stands for Continuation Information for a Patent. A patent might refer to other patents or applications. So we use the CIP table to keep track of the related information of the referred applications and patents.

There are three attributes in the CIP table: patentNum, priorAppNo and priorPatNo. PatentNum and PriorAppNo combine the primary key. The foreign key cip.patentNum references patent.patentNum.

Because an application might not be able to become a patent, so the attribute PriorPatNo is an optional attribute in the CIP table.

Field Name	Description	Type	Primary Key	Foreign Key	Required?
priorAppNo	The application number of the referred application	NUMBER	Yes	No	Yes
priorPatNo	The patent number of the referred patent	NUMBER	No	No	No
patentNum	Patent number	NUMBER	Yes	Yes	Yes

Table 4. CIP Table Design

- **Claim Table**

A patent has an abstract, which includes one or more claims. Claims can be dependent or independent. If a claim has mentioned other claims in its claim content, then it is a dependent claim. For example, claim 1 doesn't mention other claims, so it is independent. Claim 2 is dependent if it mentions " as claimed in claim 1".

The claim table has four attributes: patentNum, claimID, content, and dependent. The primary key is combined by patentNum and claimID. The foreign key claim.patentNum references patent.patentNum.

Field Name	Description	Type	Primary Key	Foreign Key	Required?
patentNum	Patent number	NUMBER	Yes	Yes	Yes
claimID	The claim identification number	NUMBER	Yes	No	Yes
content	Content of the claim	VARCHAR (4000)	No	No	Yes
dependent	If the claim is dependent	NUMBER (1)	No	No	No

Table 5. Claim Table Design

- **Foreign Table:**

Sometimes a patent also has companions identified as patents issued in a foreign country. The foreign table is used to keep track of the country and the patent Number in that country. The foreign table has four attributes: patentNum, foreignID, fDate and country. PatentNum and ForeignID combine the primary key. The foreign key foreign.patentNum references patent.patentNum).

Field Name	Description	Type	Primary Key	Foreign Key	Required?
patentNum	Patent number	NUMBER	Yes	yes	Yes
foreignID	The foreign patent number	VARCHAR2 (15)	Yes	No	yes
fDate	The filed date of the foreign patent number	DATE	No	No	yes
country	The foreign country of the patent	VARCHAR2 (15)	No	No	Yes

Table 6. Foreign Table Design

- **Assignee Table and Assign Table**

We keep track of the assignees of the patents by the assignee table. Since one patent might have more than one assignee and one assignee might be assigned more than one patent, we use the assign table to build the relationship between the patent table and the assignee table. The assignee table has six attributes: name, orgType, place, country, assigneeCode and assigneeID, the primary key is assigneeID. The assign table has only two attributes: patentNum and assigneeID and they are combined as the primary key. The foreign key assign.patentNun references the patent.patentNum and the foreign key assign.assigneeID references the assignee.assigneeID.

Field Name	**Description**	**Type**	**Primary Key**	**Foreign Key**	**Required?**
assigneeID	The identified number given to the specific assignee (please refer to the text)	NUMBER	Yes	No	Yes
assigneeCode	Three cases: 1.Some assignee has assignee code. 2.Some assignee doesn't have assignee code.	NUMBER	No	No	Yes
Name	The name of the assignee	VARCHAR (200)	No	No	Yes
orgType	The organization type of the assignee, there are four types: 1.company /corporation 2.government 3.university 4.individual	VARCHAR (25)	No	No	Yes
Place	The assignee's state	VARCHAR (30)	No	No	Yes
country	The assignee's country	CHAR (25)	No	No	Yes

Table 7. Assignee Table Design

Field Name	Description	Type	Primary Key	Foreign Key	Required?
patentNum	Patent number	NUMBER	Yes	Yes	Yes
assigneeID	The ID of the assignee (refer to Table 7.)	NUMBER	Yes	Yes	Yes

Table 8. Assign Table Design

- **ReAssign Table:**

Sometimes a patent is reassigned. The reassign table is used to keep track of the name of the re-assignee. The reassign table has only two attributes, patentNum and reAssignee.

The primary key is the combined patentNum and reAssignee. The foreign key reAssign.patentNum references patent.patentNum.

Field Name	Description	Type	Primary Key	Foreign Key	Required?
patentNum	Patent number	NUMBER	Yes	Yes	Yes
reAssignee	The name of the company reassigned to the patent	VARCHAR (255)	Yes	No	Yes

Table 9. ReAssign Table Design

3. Data Piping Design

After we successfully created the tables in Oracle8*i* database server, how to pipe more than 1600 patents into our database became the most critical issue. The 1600 patents are

in ASCII text formats, and one file includes 25 to 100 patents depending on how the research assistant captured them from the Dialog database. We decided to write a PERL script to automate the piping process and there are a couple of reasons about why we chose PERL as follows:

- Stands for Practical Extraction and Report Language.
- Freely available, and already installed at Ruby
- One of the most portable programming languages available today
- Great features in text processing like pattern matching

The pattern match feature in PERL greatly enhanced the speed of the automatic data piping script. The idea behind the automatic data piping script is really easy. For one single patent, whenever the program reads “Utility”, we will get the data between the “Utility” and “PATENT NO.”, and place the data into the pre-defined variable “$title”. Through this method, we are able to get all the variables we need, and generate Structure Query Language (SQL) based on the variables. Whenever the program reads the next patent, we defined all the existing variables to be empty or zero, and began our next patent’s capturing data. Please refer to appendix A-2 for the whole PERL script.

The biggest challenging in the data piping design is the pilot study I did on piping the very large documents into the CLOB data type. Compared with general varchar2 data types, CLOB is very special. You have to use the following programming environments to operating on LOBs:

- Using the DBMS_LOB Package for writing with LOBs
- Using the Oracle Call Interface (OCI) with LOBs

- Using C++ to work with LOBs
- Using COBOL to work with LOBs
- Using Visual Basic to work with LOBs
- Using Java to work with LOBs

After a quick comparison, I decided to use the first solution—using Oracle's specific programming language PL/SQL and its package DBMS_LOB to insert and display data from CLOBs. The main reason is that the DBMS_LOB package could provide as many features as we would like and I am confident about PL/SQL.

```
INSERT INTO test_table VALUES(4933334, empty_clob());

DECLARE
textinfo clob := empty_clob();
lengthOfText integer := 0;
buffer varchar2(10000) := '<text document goes here>';
BEGIN
   SELECT introPage INTO textinfo FROM test_table WHERE patentNum=4167589
FOR UPDATE;

   dbms_lob.write(textinfo, length(buffer), 1, buffer);
   UPDATE test_table SET introPage = textinfo WHERE patentNum=4167589;
   COMMIT;
```

END;

Script 1. Insert data into CLOB data type using PL/SQL

4. Web Database Architecture Design

Java Server Pages could be designed with a series of different architectures, but I decided to use the client-side approach and designed the following three-tier architecture.

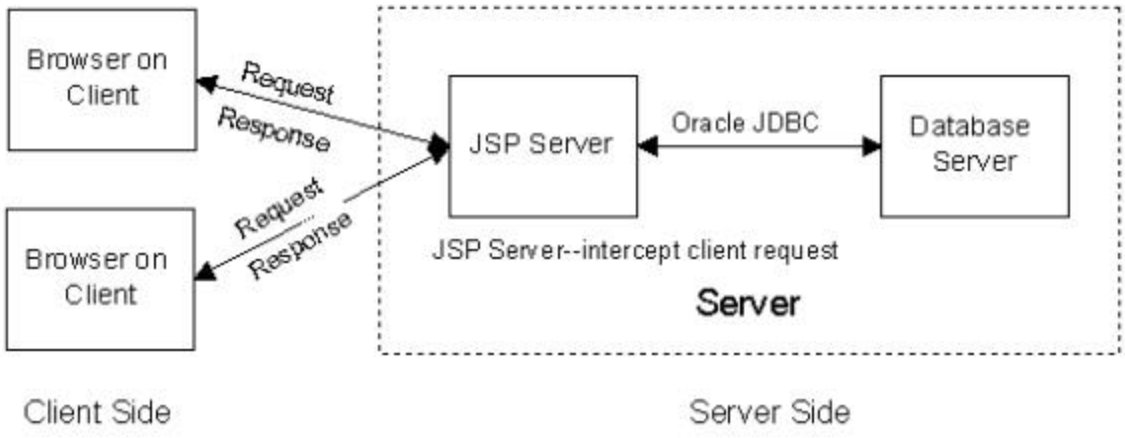


Fig 2. Three-tier Architecture Design

On the client side, whenever users request a new dynamic page, JSP server will parse the data from the users' input and build a JDBC connection and communicate with the Oracle8*i* server. After getting the data from the Oracle8*i* database server, JSP server will provide the query data in html format and display on the users' browsers. Avedal, Holden & Zeiger (2000) pointed out "The advantage of such an approach is that it is simple to program, and allows the page author to generate dynamic content easily, based upon the

request and the state of the resources.” But the potential problem of this approach is that the JSP server has to build JDBC connection to the database server in each single request by the clients. So this architecture doesn’t scale up very well for a significant number of simultaneous clients since there will be large amount of requests from the client sides.

Chapter 5. Implementation and Results

1. Oracle8*i* *inter*Media Implementation

a) Create Index

After the tables are created and data is piped into the database, I used Oracle8*i* interMedia to index some text data types. Building the index is the necessary requirement before running the CONTAINS query on the text data.

First of all, I used the create index SQL command and created an index on the content field in the claim table:

```
CREAT INDEX claim_index ON claim(content)
indextype is ctxsys.context;
```

Second, I ran the following SQL to rebuild the index. If you failed to rebuild the index, the new created index won't work.

```
ALTER INDEX claim_index REBUILD;
```

There are some issues related to indexing:

- View indexing is not allowed in Oracle8*i*

- Only one column is allowed in the column list. This column must be one of the following types: CHAR, VARCHAR, VARCHAR2, LONG, LONG RAW, BLOB, CLOB, BFILE
- Date, number, and nested table columns cannot be indexed
- In order to be indexed, the table must also have a primary key constraint.
- If a syntax error occurs in the create index statement, the index is still created. Before you reissue the corrected statement, you must drop the failed index first.
- You could use the alter index command to alter your index if an error occurs during actual indexing (e.g. you run out of tablespace)
- Once the index is created, any export will include the index definition. At import time, imp will re-create the index by issuing the create index statement.

Third, if you delete or update some documents on the claim table, the old information must be removed from the index so that queries will no longer hit it. Usually you built the index before you update or delete some documents, which might bring some potential problem in querying the claim table. Oracle8*i* *inter*Media is smart enough to mark the old document content as invalid and does not touch the index. But this still leaves the old information in the index, and takes up space in the index table. You could solve these potential problems by running optimization. Optimization has two modes: FAST and FULL. FAST optimization targets fragmentation only:

```
ALTER INDEX claim_index REBUILD ONLINE parameters ('optimize fast');
```

FAST optimization runs through the whole index table and glues fragmented rows together, which will reduce the number of rows per index table. FULL optimization does both defragmentation and garbage collection, which removes the old information left over after document invalidation. Of course, FULL optimization is more powerful, but it takes much longer time.

ALTER INDEX claim_index REBUILD ONLINE parameters ('optimize full');

2) CONTAINS Function

Oracle8*i* *inter*Media offers a broader range of text-searching capabilities than the exact match of words, by using the CONTAINS function. It provides much more capabilities than the LIKE operator's pattern matching.

- **Exact matches of a word or phrase**

In this query, it will query the claim table for all patent numbers whose claim content includes the word 'supercritical carbon dioxide'. The CONTAINS function has two parameters, the first one is the column name you want to query, the second one is the keyword you would like to search on the column name. The CONTAINS function prompts the *inter*Media server process to check the text index for the "content" column. If the word 'supercritical carbon dioxide' is found in the content column's text index, then a score greater than zero is returned by the database, and the matching patentNum value is returned.

```
SELECT      patentNum
FROM        claim
WHERE       CONTAINS(content, 'supercritical carbon dioxide')>0;
```

The '>' sign in the CONTAINS function is called a threshold operator. The threshold analysis compares the score—the internal score *inter*Media calculated when text search was performed with the specified threshold value, which is 0 here. Score values for individual searches ranges from 0 to 10 for each occurrence of the search string within the text. Furthermore, you could display the score as part of your query.

```
SELECT          patentnum, claimid, SCORE(10)
FROM            claim
WHERE           contains(content, 'supercritical', 10)>0
ORDER BY        SCORE(10) desc;
```

In the preceding query, the CONTAINS function has three parameters. Besides the column name and keyword, it includes a label ('10') for the text search operation performed. The SCORE function will display the score of the text search associated with that label. Also, if you would like order by SCORE, you could use order by clause.

But how are the scores in the CONTAINS function calculated by Oracle8*i* *inter*Media? Scores can be any number, and the scoring is relative, which means that scores are not comparable across indexes, or even across different queries on the same index. From

Loney & Koch's Oracle8*i* The Complete Reference (2000), I know that score for each document is computed using the standard Salton formula:

3f(1+log(N/n))

Where f is the frequency of the search term in the document, N is the total number of rows in the table, and n is the number of rows, which contain the search term. This is converted into an integer in the range 0 - 100.

- **Exact matches of multiple words, using Boolean logic to combine searches:**

The following query will retrieval any content from the claim table if it contains both 'supercritical' and 'carbon dioxide'. It doesn't make a difference if carbon dioxide appears before supercritical. Of course, you could use more than one 'AND' operator in the CONTAINS second parameter. You could also use ampersand (&) to replace 'AND', but it is not recommended, because Oracle usually treats ampersand (&) as an indicator of a variable and prompt you for the value of the variable.

```
SELECT    content
FROM      claim
WHERE     CONTAINS(content, 'supercritical AND carbon dioxide')>0;
```

The OR operator will return a big range. If the content contains either 'supercritical' or 'carbon dioxide' and meets the defined threshold, the record will be returned. In other

words, records will be returned only if CONTAINS(content, ‘supercritical’)>0 and CONTAINS(content, ‘carbon dioxide’)>0. The vertical line (|) is the same as ‘OR’ in the CONTAINS function.

```
SELECT    content
FROM      claim
WHERE     CONTAINS(content, ‘supercritical | carbon dioxide’)>0;
```

The vertical line (|) is the same as ‘OR’ in the CONTAINS function. In our test Oracle8*i* database, the AND query returns only 2 records while the OR query returned 37 records.

The ACCUM(accumulate) operator provides another method for combining searches. It adds together the scores of the individual searches and compares the accumulated score to the threshold value. It uses comma(,) as the symbol for ACCUM. For example:

```
SELECT    content
FROM      claim
WHERE     CONTAINS(content, ‘supercritical ACCUM carbon dioxide’)>0;
```

It is very reasonable to get the same 37 records as the OR query.

MINUS queries are used to subtract the scores from multiple searches before comparing the result to the threshold score. In the example below, the MINUS operator subtracts the

score of ‘carbon dioxide’ search from the score of ‘supercritical’ search. The symbol ‘-’ could replace the MINUS operator.

```
SELECT    content
FROM      claim
WHERE     CONTAINS(content, ‘supercritical - carbon dioxide’)>0;
```

We got 28 records, which excludes 9 records out.

We could also use parentheses to clarify the logic within our search criteria and build complex queries like below. In this case, records will be returned if either CONTAINS(content, ‘supercritical’)>0 or CONTAINS(content, ‘carbon AND dioxide’)>0.

```
SELECT    content
FROM      claim
WHERE     CONTAINS(content, ‘supercritical  OR (carbon AND dioxide)’)>0;
```

- **Search based on how close words are to each other.**

Oracle8*i* *inter*Media provides proximity search capabilities to perform a text search based on how close terms are to each other within the searched table column. It will return a high score for words that are next to each other and a lower score for words that are far

away from each other. You could replace NEAR with its equivalent symbol—semicolon(;).

```
SELECT      patentno, SCORE(10), content
FROM        claim
WHERE       CONTAINS(content, 'fluid  NEAR carbon dioxide', 10)>0;
```

Using wildcards during searches:

The Like query also performs the same wildcards functions using the special character '%' or "_'. The percent sign can be pattern matched with one or multiple characters, while underscore can be pattern matched for only one character. For example, the following query will search for all text matches starting with the characters 'contain'.

```
SELECT      patentno, SCORE(10), content
FROM        claim
WHERE       CONTAINS(content, 'contain%', 10)>0;
```

However, wildcards are not as powerful as some other methods of *inter*Media like fuzzy matches, words stems and SOUNDEX matches.

- **"Fuzzy" matches of words**

Based on Loney & Koch's Oracle8*i* The Compete Reference (2000), "Fuzzy match is an expansion technique designed to find words close in form, such as mistyped or mis-OCR'ed versions." The Fuzzy Match attribute is set to the default type of fuzzy match.

Oracle8*i* *inter*Media could limit the fuzzy expansion to the best matches. The fuzzy score is a measure of how close the expanded word is to the query word. Setting up the fuzzy score will prevent the scores below from being retrieved. The following query will return the patent whose claim content has fuzzy match as the word CONTAIN.

```
SELECT    patentno, content
FROM      claim
WHERE     CONTAINS(content, '?contain')>0;
```

Besides the fuzzy match attribute, there is a STEMMER attribute. Stemming is an expansion of a word to different forms. The stem expansion of CONTAIN might include CONTAIN, CONTAINING, CONTAINED. The following example will return all claim contents that include the stem expansion of CONTAIN.

```
SELECT    patentno, content
FROM      claim
WHERE     CONTAINS(content, '$contain')>0;
```

2. Java Server Pages Implementation

As we decided to take the “platform independence” and “separate the login from design” advantage of Java Server Pages, I installed the Tomcat JSP engine on the Topaz UNIX box after I downloaded the jakarta-tomcat-3.2.1.tar file from Apache web server’s official website. After setting up some system environment variables like CLASSPATH and JAVA_HOME, we are able to run the Tomcat JSP server on http://topaz.ils.unc.edu:8888/.

The second step after installing the Tomcat JSP server is to build the Oracle8*i* Java Database Connection, so that the JSP server could communicate with the Oracle8*i* server. This step was much more challenging than installing the Tomcat JSP server, although the Oracle8*i* server is running on the same Solaris box as the Tomcat JSP server. After confirming with the UNIX system administrator, I know that we are running Oracle8*i* 8.1.6 and Java Development Kit (JDK) 1.1, which excludes many JDBC drivers. But I still had to choose a JDBC driver from the following:

- JDBC OCI Driver 8.1.6
- JDBC Thin Driver 8.1.6
- JDBC Thin Server-side Driver 8.1.6
- JDBC Server-side Internal Driver 8.1.6

JDBC OCI Driver and JDBC Thin Driver is client side JDBC. Because Oracle Call Interface (OCI) 8.1.6 is a pre-requisite for JDBC OCI Driver, and we already installed Oracle OCI on Topaz, we decided to use JDBC OCI Driver 8.1.6 to enable the communication between the Oracle server and the JSP server. I used the "divide and conquer" strategy in the web interface design tasks. First, I set up the JDBC system environment, and then I verified the correct functionality of Oracle JDBC using a simple Java application.

Third, I started the interface prototype developing by using JSP without wasting time to figure out either the JDBC part or other part of my JSP script is broken. Because I have learned Active Server Pages before, I spent much less time in learning JSP than setting up JSP system requirements. I found out that JSP is really easy to learn and carries all the advantages of Java!

3. System Prototype Results

Users could use Boolean search to search on the keyword in their selected section. For example, users could do a search "supercritical carbon dioxide" on a claim's content field and "carbon dioxide" on the patent's title at the same time. The relative searching scores, patent number, and title will be retrieved in html table format and mark the search criteria

before the table. If users would like to view more detailed information about each patent retrieved, they could click on the patent number link and almost all information about the clicked patent will be retrieved. It includes the patent's title, inventor's information, assignee, application number, patent's abstract and so on.

If users already know which patent number they want, they could go to the exact match number match page to get the information about the patent. Based on users' request, multiple patent number search, as long as the numbers are separated by one or more than one space, is also provide to enhance the users' search efficiency.

Chapter 6. Conclusions and Recommendations

1. Conclusions

This project implemented a novel project using Oracle8*i*'s newest key option—Oracle8*i* *inter*Media, which is easy to learn and performs complex searching ability from Boolean to fuzzy match, and SOUNDEX search. This project also approaches the newest web database technology—Java Server Pages and marries it with Oracle8*i* *inter*Media to provide fast and accurate structured and unstructured data retrieval to the users. The relative score feature provided by Oracle8*i* *inter*Media turns out to be very important for users to judge patents' relevancy.

The project has also evaluated the Oracle8*i* *inter*Media's functionality and studied how *inter*Media works with various queries and data types especially the varchar2(4000) and CLOB data. It focuses on how to use *inter*Media to retrieve relative data and provide the data in html formats using Java Server pages. Furthermore, it also evaluates the three-tier web architecture that uses the JSP as the middleware and Oracle8*i* as the backend.

2. Recommendations

If we have more time, I strongly recommend implementing another project using some search engine like WAIS and the current Oracle8*i* database—assuming that Oracle8*i* doesn't provide *inter*Media function. We could evaluate the project with our current project's performance from:

- Retrieved data relevancy and accuracy.
- Retrieval speed (single client, and multi-client simultaneously). And what factors influence the retrieval speed.
- Retrieval functionality like Boolean search, fuzzy match, scoring, and so on.
- Feasibility to maintain, and extensibility.

Besides the above study, I also recommend using eXtensible Markup Language (XML) and metadata to mark up the unstructured data based on a widely accepted Document Type Definition (DTD). I believe, this will greatly enhance our database's communication and data exchange with other related patent database or chemistry related database plus the accuracy of information retrieval.

Bibliography

1. Loney, K. & Koch G. (2000). Oracle8*i* The Complete Reference
2. Koch, G. & Loney, K. (1997). Oracle8 The Complete Reference
3. Wall, L., Orwant J. & Christiansen, T. (2000). Programming Perl
4. Java Server Pages: Dynamically Generated Web Content. Retrieved from Sun's official
5. Java Server Pages Homepage, http://java.sun.com/products/jsp/index.html.
6. Blair, D. (1984). Information Retrieval on Large Documents. Communications of the ACM, April 1984.
7. Avedal K., Holden M. & Zeiger S. (2000). Professional JSP.
8. Oracle *inter*Media Text Services. Retrieved from Oracle Technology Network (OTN) web site, http://technet.oracle.com/products/intermedia/htdocs/text_techover.html.
9. Oracle Application Developer's Guide Release 8.0. Retrieved from Oracle Technology Network (OTN) web site, http://technet.oracle.com/doc/server.804/a58241/toc.htm.
10. Oracle JDBC Frequently Asked Questions. Retrieved from Oracle Technology Network (OTN) web site, http://technet.oracle.com/tech/java/sqlj_jdbc/htdocs/jdbc_faq.htm.
11. Oracle8*i* SQL Reference Release 8.1.5. Retrieved from Oracle Technology Network (OTN) web site, http://technet.oracle.com/doc/server.815/a67779/toc.htm.

12. Oracle8*i* *inter*Media Text Reference Release 8.1.5. Retrieved from Oracle Technology Network (OTN) web site, http://technet.oracle.com/doc/inter.815/a67843/toc.htm.
13. Oracle8*i* JDBC Developer's Guide and Reference Release 8.1.5. Retrieved from Oracle Technology Network (OTN) web site, http://technet.oracle.com/doc/java.815/a64685/toc.htm

Appendix A—Sample Patents Introduction Page

Utility
ANTIBIOTIC COMPOSITION
[AND A PHARMACEUTICALLY ACCEPTABLE, WATER SOLUBLE ALKALI METAL CARBONATE;
STORAGE STABILITY]

PATENT NO.: 4,933,334
ISSUED: June 12, 1990 (19900612)
INVENTOR(s): Shimizu, Hisayoshi, Osaka, JP (Japan)
Mikura, Yasushi, Osaka, JP (Japan)
Doi, Yasuo, Hyogo, JP (Japan)
ASSIGNEE(s): Takeda Chemical Industries, Ltd , (A Non-U.S. Company or Corporation), Osaka, JP (Japan)
[Assignee Code(s): 82624]
EXTRA INFO: Expired, effective June 15, 1994 (19940615), recorded in O.G. of August 23, 1994 (19940823)
APPL. NO.: 7-274,977
FILED: November 22, 1988 (19881122)
PRIORITY: 62-308350, JP (Japan), December 4, 1987 (19871204)
FULL TEXT: 707 lines

ABSTRACT

An antibiotic composition which comprises 7 beta -[(Z)-(5-amino-1,2,4-thiadiazol-3-yl)-2(Z)-methoxyiminoacetamide]-3(1-imidazo[1,2-b]pyridazinium)-methyl-3-cephem-4-carboxylate hydrochloride and a pharmaceutically acceptable water-soluble basic substance, which is stable in storage and improved in solubility as well as free of local actions or hemolytic action.

What we claim is:

1. An antibiotic composition which comprises an effective antibacterial amount of 7 beta -[(Z)-2-(5-amino-1,2,4-thiadiazol-3-yl)-2(Z)-methoxyiminoacetamido]-3-(1-imidazo[1,2-b]pyridazinium)methyl]-3-cephem-4-carboxylate hydrochloride and a pharmaceutically acceptable [water-soluble basic substance] alkali metal carbonate in an equivalent ratio of about 1:1.2 to about 1:3.0 relative to said hydrochloride.

2. An antibiotic composition as claimed in claim 1 wherein the alkali metal carbonate is sodium carbonate.

3. An antibiotic composition as claimed in claim 1 wherein the alkali metal carbonate is sodium hydrogen carbonate.

4. An antibiotic composition as claimed in claim 1 which comprises the pharmaceutically acceptable water-soluble basic substances in an amount of about 1.4 to 2.0 equivalent relative to one equivalent of 7 mu -[(Z)-2-(5-amino-1,2,4-thiadiazol-3-yl)-2(Z)-methoxyiminoacetamido]-3-(1-im idazo[1,2-b]pyridazinium)methyl]-3-cephem-4-carboxylate hydrochloride.

Appendix A—SQL Generator

```
#! /usr/local/bin/perl

#########################################################
#This is the SQL generator script developed by Lin
#Sun using Perl Programming Language. This script will
#parse any patent introduction page and generate the
#SQL in the screen or output file.
#Usage: genSQL.pl <inputfile>
#       or genSQL.pl <inputfile> ><outputfile>
#The first command will print out the output on
#the scrren, while the second command will pipe
#the output into your specified output file.
#########################################################

#########################################################
#The following process are for inventor and assignee
#only. Because these two fields are very special. For
#example, an inventor might invent many patents. Since
#the input file doesn't provide us any unique
#identification number of inventor, we have to store
#inventor's first name, last name, middle initial
#into a record.txt file, and whenever process a new
#patent introduction page, we will compare the inventor
#with the inventor in the record.txt file to judge if
#it is a new inventor or old inventor. Same thing
#happens with assignee
#########################################################

#open inventor's record file.
do 'mylib.pl';
&init_table;
open(RECORD, ">>record.txt");

@nameList=keys %table;
$maxSeqNum=@nameList;

#open assignee's record file.
do 'mylib_assignee.pl';
```

```
&init_table_assignee;
open(RECORD2, ">>record_assignee.txt");

@nameList2=keys %table2;
$maxSeqNum2=@nameList2;


#########################################################
#Using a while loop here to read every line of one
#patent introduction page. I set up many flags to
#locate the title part, inventor page, assignee part
#and so on. This code should be read simultaneously
#with the patent introduction page template.
#########################################################

#set Flag==0 at the beginning.
&resetFlag;
$count=0;
while(<>)
{
      #Chop the enter key and change it to space
       s/\n/ /g;
      #Chop all space larger than 1 space to 1 space only.
      s/\s{2,}/ /g;

      #Do pattern match with Utility.
      #set Flag at the beginning.
       if (/^Utility/)
       {
            if ($count)
            {
                  &processEachPatent;
                  &resetVar;
            }
            $count++;
            &resetFlag;
            $flagUtility=1;
       }

      #Do pattern match with PATENT NO.
      #set Flag at the beginning.
       elsif (/^PATENT NO.: ([0-9].*[0-9])/)
       {
            &resetFlag;

            $_=$1;
```

```
        #chop all ",".
        s/,//g;
        $patentNum=$_;
}

#Do pattern match with ISSUE.
#set Flag at the beginning.
elsif (/^ISSUE.*\(([0-9]{8})\)/)
{
        &resetFlag;
        $issueDate=$1;
}

#Do pattern match with INVENTOR.
#set Flag at the beginning.
elsif (/^INVENTOR.*: (.*)/)
{
        &resetFlag;
        $flagInventor=1;

        push(@inventors, $1);
}

#Do pattern match with ASSIGNEE.
#set Flag at the beginning.
#This one is complex because for Reassignee, they also
#use Assignee(s). So we need to judge if this is the first occurance of
#Assignee(s).
elsif (/^ASSIGNEE.*: ([a-zA-Z].*)/)
{
        #if flagReAssignee=1, then it is not the first occurance of
        #Assignee(s), so it belongs to the reAssignee apart, then push
        #into the reAssignee array.
        if ($flagReAssignee)
        {
                $reAssignee.=$_;
                #push(@reAssignee, $_)
        }
        #else, it belongs to the assignee part, so get the value
        #of the first line of Assignee.
        else
        {
                &resetFlag;
                $flagAssignee=1;
                $assignee=$1;
        }
```

```
}

#Do pattern march with Assignee Code.
#set Flag at the beginning.
elsif (/Assignee Code\(s\): ([0-9]*)/)
{
	&resetFlag;
	$assigneeCode=$1;
}

#Do pattern match with EXTRA INFO.
#set Flag at the beginning.
elsif (/^EXTRA INFO: ([a-zA-Z].*)/)
{
	&resetFlag;
	$flagExtraInfo=1;
	$extraInfo=$1;
}

#Do pattern match with POST-ISSUANCE ASSIGNMENTS.
#set Flag at the beginning.
elsif(/POST-ISSUANCE ASSIGNMENTS/)
{
	&resetFlag;
	$flagReAssignee=1;
}

#Do pattern match with APPL. NO.
#set Flag at the beginning.
elsif (/^APPL. NO.: ([0-9].*[0-9])/)
{
	&resetFlag;

	$_=$1;
	s/,//g;
	s/-//g;
	$applNum=$_;
}

#Do pattern match with FILED.
#set Flag at the beginning.
elsif (/^FILED.*\(([0-9]{8})\)/)
{
	&resetFlag;
	$filedDate=$1;
}
```

```
#Do pattern match with PRIORITY.
#set Flag at the beginning.
elsif (/^PRIORITY: (.*)/)
{
	&resetFlag;
	$flagPriority=1;
	$priority=$1;
}

#Do pattern match with CIP.
#set Flag at the beginning.
elsif (/^ This .*application.*/)
{
	&resetFlag;
	$flagCIP=1;
	$CIP=$_;
}

#Do pattern match with FULL TEXT.
#set Flag at the beginning.
elsif (/^FULL TEXT:/)
{
	&resetFlag;
}

#Do pattern match with ABSTRACT.
#set Flag at the beginning.
elsif (/ABSTRACT/)
{
	&resetFlag;
	$flagAbstract=1;
}

#Do pattern match with What we claim is.
#set Flag at the beginning.
elsif (/claim.{0,60}:/i)
{
	&resetFlag;
	$flagClaim=1;
}

#find the end of every patent.
elsif (/^[\s]{0,1}[0-9]{1,2}\/[0-9]{1,2}\/[0-9]{1,4}/)
{
	&resetFlag;
```

```
}

#find the end of whole pat file.
elsif(/### Status: Signed Off./)
{
	&resetFlag;
	$flagEnd=1;
}
elsif ($flagUtility)
{
	$utility.=$_;
}

elsif ($flagInventor)
{
	#/(.*)/;
	push(@inventors, $_);
}

elsif ($flagReAssignee)
{
	#push(@reAssignee, $_);
	$reAssignee.=$_;
}

elsif ($flagAssignee)
{
	$assignee.=$_;
}

elsif ($flagExtraInfo)
{
	$extraInfo.=$_."'||\n'";
}

elsif ($flagCIP)
{
	$CIP.=$_;
}

elsif ($flagAbstract)
{
	s/'/''/g;
	$abstract.=$_."'||\n'";
}
```

```
    elsif ($flagClaim)
    {
        if ($_ ne ' ')
        {
            s/'/''/g;
            $claim.=$_."'||\n'";
        }
    }
}
&processEachPatent;


#########################################################
#This function is to process all the variables & arrays
#we got from the while loop. Bascially, we got the
#section of variable from while loop, but we also need
#to divide the section into small detailed variables.
#########################################################

sub processEachPatent
{
    &processTitle;
    &processInventors;
    &processAssignee;
    &processExtraInfo;
    &processReAssignee;
    &processPriority;
    &processCIP;
    &processClaim;
    &showRes;
}

#########################################################
#resetVar function is crucial if the input file contains
#more than one patent, which is the general case. It will
#reset all the variables and arrays to empty or zero before
#it process next patent.
#########################################################

sub resetVar
{
    $abstract="";
    $applNum=0;
    $assignee="";
    $assigneeCode=0;
    $assigneeCopy="";
```

```
$assigneeCountry="";
$assigneeName="";
$assigneePlace="";
$assigneeKey=0;
$CIP="";
$CIPappNo=0;
$CIPpatentNo=0;
$claim="";
$claimBad=0;
$country="";
$dependent=0;
$detailCIP="";
$expired="";
$expiredDate="";
$extraInfo="";
$filedDate="";
$fName="";
$inventor="";
$issueDate="";
$len="";
$lName="";
$mInitial="";
$orgType="";
$patentNum=0;
$place="";
$priority="";
$priorityCountry="";
$priorityDate="";
$priorityNo="";
$reAssignee="";
$reAssigneeName="";
$title="";
$utility="";

#set all arrays to empty.
@assigneeCountry=();
@assigneeName=();
@assigneePlace=();
@CIPappNo=();
@CIPpatentNo=();
@country=();
@claim=();
@detailCIP=();
@fName=();
@inventors=();
@lName=();
```

```
	@mInitial=();
	@oldReAssignee=();
	@orgType=();
	@place=();
	@realInventors=();
	@reAssignee=();
	@reAssigneeName=();
}

#########################################################
#Reset all the flags to zero. This is extremely
#important when processing a signle patents or
#multi-patents. Without the correct flag setting
#the script won't be able to recognize the correct
#section.
#########################################################
sub resetFlag
{
	$flagUtility=0;
	$flagInventor=0;
	$flagAssignee=0;
	$flagExtraInfo=0;
	$flagReAssignee=0;
	$flagPriority=0;
	$flagCIP=0;
	$flagAbstract=0;
	$flagClaim=0;
	$flagEnd=0;
}

#########################################################
#Process Title: clean up special characters in
#title variable.
#########################################################

sub processTitle
{
	$_=$utility;
	s/\[.*\]//g;
	$title=$_;
}

#########################################################
#processInventors: Because we have serveral inventors
#for one patent, we used array in the while loop to
#capture each inventor information into an array.
```

```
#In this function, we divided each inventor's information
#into fname, lname, minitial, place, & country and
#store them into five arrays.
#Note: This function might not be smart enough
#when comes to very complex inventors.
#########################################################

sub processInventors
{
	foreach(@inventors)
	{
		#Judge if it is a line of new inventor or continued line of last
		#inventor based on how many commas. If larger than 3 commas, then
		#it should be a new inventor.
		if (/.*,.*,.*,.*/)
		{
			if ($inventor)
			{
				push(@realInventors,$inventor);
			}
			s/\s{2,}//g;
			$inventor=$_;
		}
		#If it is not a new inventor, adds this line to last inventor.
		else
		{
			s/\s{2,}//g;
			$inventor.=$_;
		}
	}
	push(@realInventors,$inventor);

	#Parse information get from Inventor.
	foreach(@realInventors)
	{
		#Split whenever hit ",".
		split(/,/);
		$len=@_;

		#Get the inventor's last name.
		$_=$_[0];
		#Chop the first space.
		s/^ //g;
		$lName=$_;

		#Get the inventor's first name.
```

```
$_=$_[1];
#Chop the first space.
s/^ //g;
#See if fName has middle initial in it.
if (/^([a-zA-Z]*) ([A-Z])./)
{
	$fName=$1;
	$mInitial=$2;
}
else
{
	$fName=$_;
}

#see if the last word of inventor is composed of numbers or not.
if ($_[$len-1]=~/[0-9]{4,6}/)
{
	#Get the inventor's place
	$_=$_[$len-3];
	#Chop all stuaff in "( )" and the space before "( )" if exist.
	s/ \(.*\)//g;
	#Chop the first space.
	s/^ //g;
	$place=$_;

	#Get the inventor's country.
	$_=$_[$len-2];
	s/\s{2,}/ /g;
	#Chop all stuaff in "( )" and the space before "( )".
	s/ \(.*\)//g;
	#Chop the first and the last space.
	s/^ //g;
	s/ $//g;
	$country=$_;
}
else
{
	#Get the inventor's place
	$_=$_[$len-2];
	#Chop all stuaff in "( )" and the space before "( )" if exist.
	s/ \(.*\)//g;
	#Chop the first space.
	s/^ //g;
	$place=$_;

	#Get the inventor's country.
```

```
            $_=$_[$len-1];
            s/\s{2,}/ /g;
            #Chop all stuaff in "( )" and the space before "( )".
            s/ \(.*\)//g;
            #Chop the first and the last space.
            s/^ //g;
            s/ $//g;
            $country=$_;
        }

        #Push all info. from inventor into different arrays.
        push(@lName, $lName);
        push(@fName, $fName);

        #check to see if the inventor's middle initial is blank.
        if ($mInitial eq '')
        {
            $mInitial='-';
        }
        push(@mInitial, $mInitial);

        #check to see if the inventor's place is blank.
        if ($place eq '')
        {
            $place='-';
        }
        push(@place, $place);
        push(@country, $country);
    }
}

############################################################
#processAssignee: same as inventor. Process assignee
#information and divided into assigneeName, orgType,
#place, country. The difference is we are not using
#array, because one patent usually have only one assignee.
############################################################

sub processAssignee
{
    #Make $assignee default variable and make a copy of it.
    $_=$assignee;
    s/\s{2,}/ /g;
    $assigneeCopy=$_;

    #Try to parse assignee and get assignee Name, assignee org type from it.
```

```
if (/^(.*), (\(A .*\))[\s]{0,2}, .*/)
{
	$assigneeName=$1;

	#Decide assignee's organization type.
	if ($2=~/Individual/)
	{
		$orgType="individual";
	}
	elsif ($2=~/Corporation/)
	{
		$orgType="company/corporation";
	}
	elsif ($2=~/Government Agency/)
	{
		$orgType="government";
	}
	else
	{
		$orgType="univ";
	}
}

$_=$assigneeCopy;
#Split whenever hit ",".
 split(/,/);
 $len=@_;

#Get assignee's place.
$_=@_[len-2];
#Chop all stuaff in "( )" and the space before "( )" if exist.
s/ \(.*\)//g;
 #Chop the first space.
 s/^ //g;
$assigneePlace=$_;

#Get assignee's country.
$_=@_[len-1];
#Chop all stuaff in "( )" and the space before "( )" if exist.
s/ \(.*\)//g;
 #Chop the first space.
 s/^ //g;
$assigneeCountry=$_;

 #Push all info. from assignee into different arrays.
 push(@assigneeName, $assigneeName);
```

```
        push(@orgType, $orgType);
        push(@assigneePlace, $assigneePlace);
        push(@assigneeCountry, $assigneeCountry);
}

#########################################################
#ProcessExtraInfor: Judge if expire information apprears
#in the extra info section.
#########################################################

sub processExtraInfo
{
      $_=$extraInfo;
      #Judge if Expired information occurs in extraInfo.
      if (/Expired, effective [a-zA-Z]* [0-9]*, [0-9]* \(([0-9]{8})\),.*/)
      {
            $expired=1;
            $expiredDate=$1;
      }
}

#########################################################
#processReAssignee: get assignee's name from the long
#reassignee section. Because one patent might have
#serveral reassignee, we use @reAssigneeName to store
#the names.
#Note: we have discussed on the meeting, and Gina agreed
#to capture only the reAssigneeName in this section.
#########################################################

sub processReAssignee
{
      #Split whenever hit "ASSIGNEE(s):".
      #s/^\s*//g;
      $_=$reAssignee;
      #add "\" before "(" & ")" to split.
       @reAssignee=split(/ASSIGNEE\(s\):/);
      #delete the first entry of array reAssignee, because it will be spaces.
      #shift(@reAssignee);

      #make a copy of array reAssignee, because in foreach statement, we will do
      #do some change on the array.
      @oldReAssignee=@reAssignee;

      foreach(@reAssignee)
      {
```

```
            split(/,/);
            #Get the reAssignee's name.
            $_=@_[0];
             #Chop the first space.
             s/^ //g;
             $reAssigneeName=$_;

             #push reAssignee name into reAssigneeName array.
             push(@reAssigneeName, $reAssigneeName);
      }#end of for each

      #copy oldReAssignee array back into reAssignee array.
      @reAssignee=@oldReAssignee;
}

############################################################
#processPriority: Get priorityNo, priorityCountry, and
#priorityDate information from the priority section,
############################################################

sub processPriority
{
      $_=$priority;
      s/-//g;
      if (/^(.*), (..) \([a-zA-Z]*\), .*\(([0-9]{8})\)/)
      {
            $priorityNo=$1;
            $priorityCountry=$2;
            $priorityDate=$3;
      }
}

############################################################
#processCIP: get the CIP appNo, CIP patentnumber.
############################################################
sub processCIP
{
      #I use the occurance of "which" to tell the number of CIP patent has.
      #split whenever hits "which"
      @detailCIP=split(/which/,$CIP);

      foreach $detailCIP (@detailCIP)
      {
            $CIPappNo=0;
            $CIPpatentNo=0;
```

```
            #get application Ser. No.
            #copy $detailCIP to default variable $_ every time before if statement,
            #because it every action of if statement changes the value of $_.
            $_=$detailCIP;
            if (/Ser\. No\.\s{0,1}([0-9,-]*)/i)
            {
                $_=$1;
                s/,|-//g;
                $CIPappNo=$_;
            }

            #get US. Pat. No. if it exists. There are two cases. One is that there is
            #no "No." followed "U.S. Pat.", the other has.
            #copy $detailCIP to default variable $_ every time before if statement,
            #because it every action of if statement changes the value of $_.
            $_=$detailCIP;
            if (/U\.S\. Pat\. ([0-9,]{7,})/)
            {
                $_=$1;
                s/,//g;
                $CIPpatentNo=$_;
            }

            #copy $detailCIP to default variable $_ every time before if statement,
            #because it every action of if statement changes the value of $_.
            $_=$detailCIP;
            if (/U\.S\. Pat\. No\. ([0-9,]{7,})/)
            {
                $_=$1;
                s/,//g;
                $CIPpatentNo=$_;
             }

            #push those CIP information get into arrays.
            if ($CIPappNo)
            {
                push(@CIPappNo, $CIPappNo);
                push(@CIPpatentNo, $CIPpatentNo);
            }
        }
}

############################################################
#processClaim: get the dependent and independent
#information from computer, and store the claim
#information and dependent? information into
```

```
#two arrays.
############################################################

sub processClaim
{
	$_=$claim;
	#Split whenever hit 3 space together--"   ".
	split(/\s{1,3}[0-9]{1,2}\. /);
	#change all single ' to double '', because SQL don't recognize single '.
	 $len=@_;

	for($i=1;$i<$len;$i++)
	{
		$_=@_[$i];
		#See if the claim is dependent or indendent.
		if (/claim [0-9]{1,2}/)
		{
			$dependent=1;
		}
		else
		{
			$dependent=0;
		}

		push(@claim, $_);
		push(@dependent, $dependent);
	}
}

############################################################
#method for diaplying all the result in SQL format.
############################################################

sub showRes
{
	&cleanup;

	print("/* start of a patent information */\n\n");
	print("PROMPT $patentNum\n");

	print("clear columns;\n");
	print("INSERT INTO patent\(patentNum,
			title,issueDate,abandon,abandonDate,extraInfo,appNo,
			appFiledDate,abstract\)\n");

	print("VALUES \($patentNum,'$title',
```

```
            TO_DATE\('$issueDate','YYYYMMDD'\),'$expired',
            TO_DATE\('$expiredDate','YYYYMMDD'\),'$extraInfo',$applNum,
            TO_DATE\('$filedDate','YYYYMMDD'\),'$abstract'\);\n");
print("\n");

$len=@fName;
for ($i=0;$i<$len;$i++)
{
      print("clear columns;\n");


      $nameKey=$fName[$i]."%".$mInitial[$i]."%".$lName[$i]."%"
                  .$place[$i]."%".$country[$i];
      $nameKey=~tr/A-Z/a-z/;
      $inventorID=$table{$nameKey};
      if (!$inventorID)
      {
            $maxSeqNum++;
            print RECORD "$nameKey\n";
            print RECORD "$maxSeqNum\n";
            $table{$nameKey}=$maxSeqNum;
            $inventorID=$maxSeqNum;

            print("INSERT INTO inventor\(inventorID,fName,
                  lName,mInitial,place,country\)\n");

            print("VALUES\($inventorID,'$fName[$i]',
                  '$lName[$i]','$mInitial[$i]','$place[$i]','$country[$i]'\);\n\n");
      }
      print("INSERT INTO invent\(patentNum,inventID,inventorID\)\n");
      print("VALUES\($patentNum,invent_seq.NEXTVAL,$inventorID\);\n\n");
}

#########################################################################
#generate SQL for assignee and assign table
#We made a couple of decisions here about the generation of AssigneeID based on
#the special cases of assignees.
#
#     1. If the assigneeCode=68000, which means the inventors are the assignees,
#          assigneeID=0. In this case, we won't insert any data into assignee
#        table. But we need to insert assigneeID, and patentNum into assign
#        table.
#
#     2. Else if the assignee has assigneecode bigger than 0(not equal to
#        68000), we will insert the code into our record_assignee.txt(if it
#        doesn't exist) and assign an assigneeID to it. If the assigneeCode
```

```
#       exists in the record_assignee.txt, then get the assigneeID from it.
#
#     3. Else, we will insert assigneeName, orgType, place, country into the
#       record_assignee.txt(if it doesn't exist) and assign an assigneeID to it.
#########################################################################
print("clear columns;\n");

 #$assigneeID=$table2{$assigneeKey};

if ($assigneeCode==68000)
{
      print("INSERT INTO assign\(patentNum,assigneeID\)\n");
      print("VALUES \($patentNum,0\);\n\n");
}
else
{
      if($assigneeCode>0)
      {
            #$assigneeKey is for the second case mentioned above.
            $assigneeKey=$assigneeCode;

            $assigneeID=$table2{$assigneeKey};
            if (!$assigneeID)
            {
                  $maxSeqNum2++;
                  print RECORD2 "$assigneeKey\n";
                  print RECORD2 "$maxSeqNum2\n";
                  $table2{$assigneeKey}=$maxSeqNum2;
                  $assigneeID=$maxSeqNum2;

                  print("INSERT INTO assignee\(assigneeID,assigneeCode,name,
                        orgType,place,country\)\n");

                  print("VALUES\($assigneeID,$assigneeCode,'$assigneeName',
                        '$orgType','$assigneePlace','$assigneeCountry'\);\n\n");
            }
      }
      else
      {
            #$assigneeKey is for the third case mentioned above.
            # We replace all the uppercase with lower case.
            $assigneeKey=$assigneeName."%".$orgType."%".
                              $assigneePlace."%".$assingeeCountry;
            $assigneeKey=~tr/A-Z/a-z/;

            $assigneeID=$table2{$assigneeKey};
```

```
            if (!$assigneeID)
            {
                $maxSeqNum2++;
                print RECORD2 "$assigneeKey\n";
                print RECORD2 "$maxSeqNum2\n";
                $table2{$assigneeKey}=$maxSeqNum2;
                $assigneeID=$maxSeqNum2;

                print("INSERT INTO assignee\(assigneeID,assigneeCode,name,
                        orgType,place,country\)\n");

                print("VALUES\($assigneeID,$assigneeCode,'$assigneeName',
                        '$orgType','$assigneePlace','$assigneeCountry'\);\n\n");
            }
        }
        print("clear columns;\n");
        print("INSERT INTO assign\(patentNum,assigneeID\)\n");
        print("VALUES \($patentNum,$assigneeID\);\n\n");
}

$lenClaim=@claim;
for ($i=0;$i<$lenClaim;$i++)
{
        if ($claim[$i] eq "'||\n"')
        {
            $claimBad++;
        }
        else
        {
            print("clear columns;\n");
            $claimNo=$i+1-$claimBad;

            print("INSERT INTO claim\(claimID,content,dependent,patentNum\)\n");
            print("VALUES \($claimNo,'$claim[$i]',$dependent[$i],
                    $patentNum\);\n\n");
        }

}

if ($priorityCountry ne '')
{
        print("clear columns;\n");

        print("INSERT INTO foreign\(foreignID,fDate,country,patentNum\)\n");
        print("VALUES \('$priorityNo',TO_DATE\('$priorityDate','YYYYMMDD'\),
                    '$priorityCountry',$patentNum\);\n\n");
```

```
	}

	$lenreA=@reAssigneeName;
	for ($i=0;$i<$lenreA;$i++)
	{
		print("clear columns;\n");

		print("INSERT INTO reAssign\(patentNum,reAssignee\)\n");
		print("VALUES \($patentNum,'$reAssigneeName[$i]'\);\n\n");
	}

	$lenCIP=@CIPappNo;
	for ($i=0;$i<$lenCIP;$i++)
	{
		print("clear columns;\n");

		print("INSERT INTO CIP\(priorAppNo,priorPatNo,patentNum\)\n");
		print("VALUES \($CIPappNo[$i],$CIPpatentNo[$i],$patentNum\);\n\n");
	}

	print("/* end of a patent information */\n\n\n");
}

#########################################################
#cleanup the code, especially the special characters.
#########################################################
sub cleanup
{
	#trim spaces at the beginning or end of $title.
	$_=$title;
	s/^\s{1,3}//g;
	s/\s{1,5}$//g;
	s/'/''/g;
	$title=$_;

	#trim spaces at the beginning or end of $assigneeName.
	 $_=$assigneeName;
	 s/^\s{1,3}//g;
	 s/\s{1,5}$//g;
	 s/'/''/g;
	 s/\&/AND/g;
	 $assigneeName=$_;

	#trim spaces at the beginning or end of $orgType.
	$_=$orgType;
	s/^\s{1,3}//g;
```

```
        s/\s{1,5}$//g;
        $orgType=$_;

        #trim spaces at the beginning or end of $assigneePlace.
        $_=$assigneePlace;
        s/^\s{1,3}//g;
        s/\s{1,5}$//g;
        $assigneePlace=$_;

        #trim spaces at the beginning or end of $assigneeCountry.
        $_=$assigneeCountry;
        s/^\s{1,3}//g;
        s/\s{1,5}$//g;
        $assigneeCountry=$_;

        #trim spaces at the beginning or end of $priorityNo.
        $_=$priorityNo;
        s/^\s{1,3}//g;
        s/\s{1,5}$//g;
        $priorityNo=$_;

        #trim spaces at the beginning or end of $priorityCountry.
        $_=$priorityCountry;
        s/^\s{1,3}//g;
        s/\s{1,5}$//g;
        $priorityCountry=$_;
}
```

Appendix B—Java Server Pages Web Interface

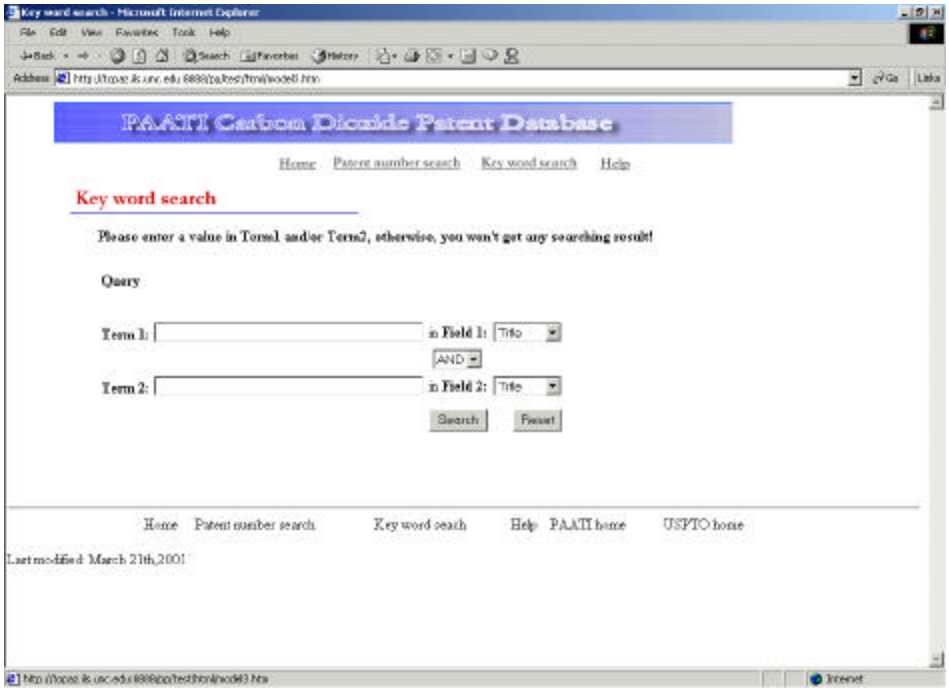
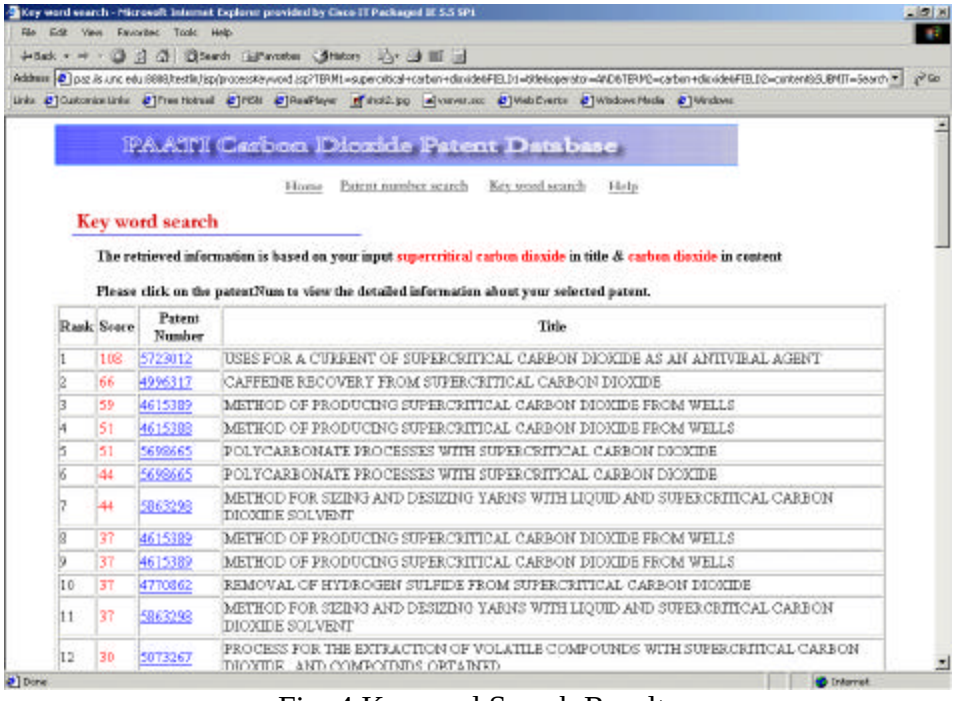


Fig. 3 Keyword Search



Rank	Score	Patent Number	Title
1	108	5723012	USES FOR A CURRENT OF SUPERCRITICAL CARBON DIOXIDE AS AN ANTIVIRAL AGENT
2	66	4996317	CAFFEINE RECOVERY FROM SUPERCRITICAL CARBON DIOXIDE
3	59	4615389	METHOD OF PRODUCING SUPERCRITICAL CARBON DIOXIDE FROM WELLS
4	51	4615388	METHOD OF PRODUCING SUPERCRITICAL CARBON DIOXIDE FROM WELLS
5	51	5698665	POLYCARBONATE PROCESSES WITH SUPERCRITICAL CARBON DIOXIDE
6	44	5698665	POLYCARBONATE PROCESSES WITH SUPERCRITICAL CARBON DIOXIDE
7	44	5863293	METHOD FOR SIZING AND DESIZING YARNS WITH LIQUID AND SUPERCRITICAL CARBON DIOXIDE SOLVENT
8	37	4615389	METHOD OF PRODUCING SUPERCRITICAL CARBON DIOXIDE FROM WELLS
9	37	4615389	METHOD OF PRODUCING SUPERCRITICAL CARBON DIOXIDE FROM WELLS
10	37	4770862	REMOVAL OF HYDROGEN SULFIDE FROM SUPERCRITICAL CARBON DIOXIDE
11	37	5863293	METHOD FOR SIZING AND DESIZING YARNS WITH LIQUID AND SUPERCRITICAL CARBON DIOXIDE SOLVENT
12	30	5073267	PROCESS FOR THE EXTRACTION OF VOLATILE COMPOUNDS WITH SUPERCRITICAL CARBON DIOXIDE AND COMPOUNDS OBTAINED

Fig. 4 Keyword Search Results

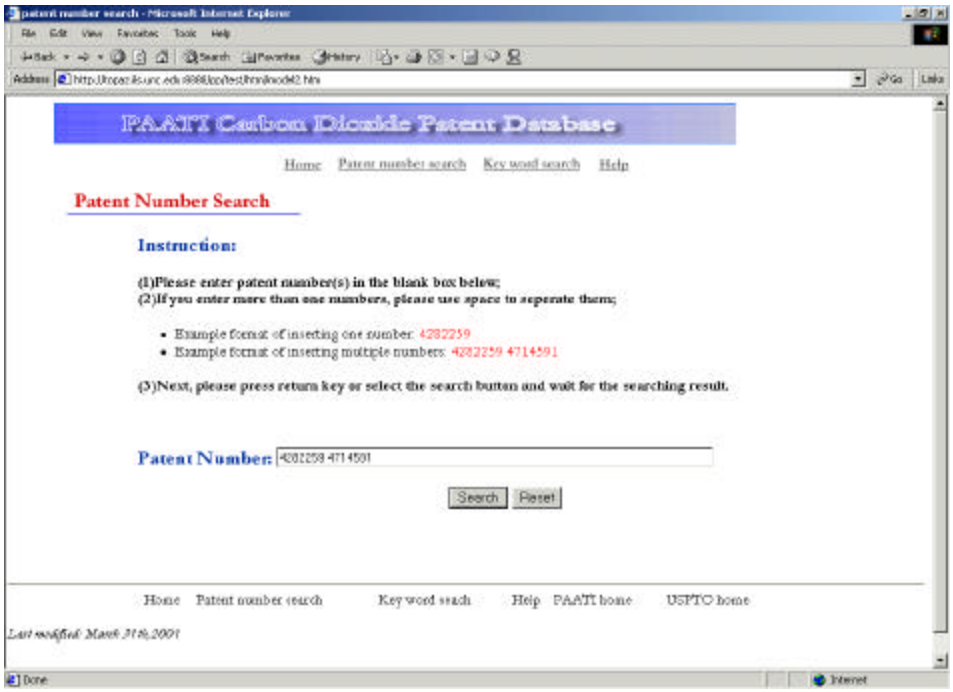


Fig. 5 Patent Number Search

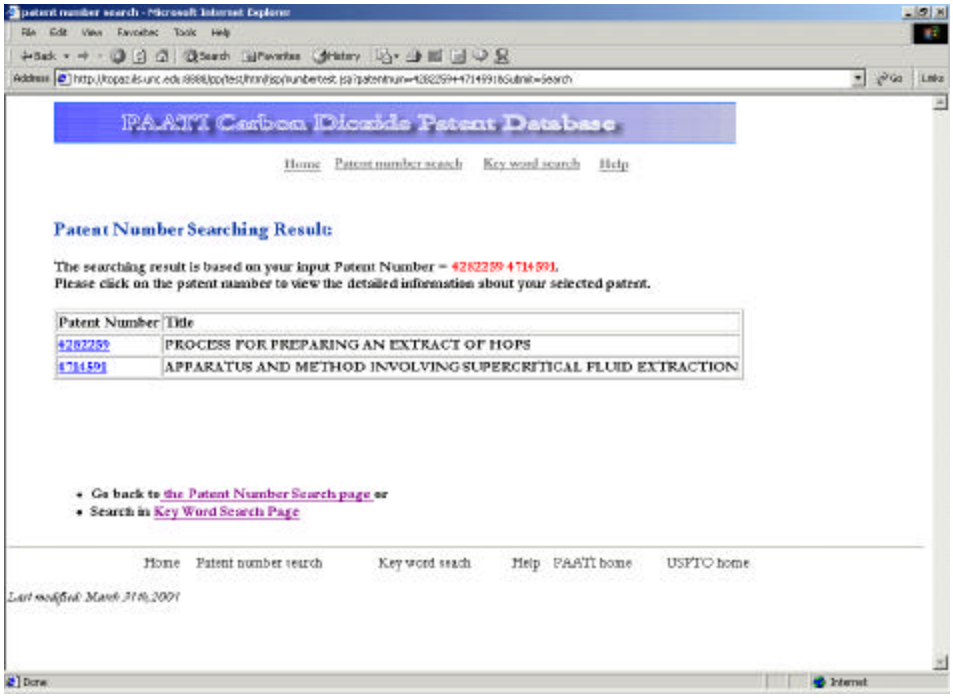


Fig. 6 Patent Number Search Results

Appendix B— JSP: Keyword Search

```
<%
/*
processKeyword.jsp
Search keyword on patent title, abstract, claim from Patent Project Oracle 8i database
Call using http://..../processKeyword.jsp
*/
%>


<% // ---------- Begin JSP Directives ---------- %>
<%@ page import="javax.servlet.*" %>
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.sql.*" %>
<%@ page language="java" %>

<%
// Begin variable definitions
String inputTerm1 = ""; // User's term1 input
String inputTerm2 = ""; //User's term2 input
String inputField1 = ""; //User's field1 select
String inputField2 = ""; //User's field2 select
String inputOperator = ""; //User's operator input

Statement stmt = null;
StringBuffer qry = new StringBuffer(1024);


/* Make connection to the database */
/* Set username and password for the database - blank in this case */
String db="sils";
String user= "pp_4nsf";
String passwd = "*******";


/* Declare Connection and Result Set variables */
Connection conn = null;
ResultSet rs = null;


try
{
   /* Check we can find the database driver */
```

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

conn = DriverManager.getConnection
          ("jdbc:oracle:oci8:@" + db, user, passwd);

/*Get the input values*/
inputTerm1 = request.getParameter("TERM1"); // User's term1 input
inputTerm2 = request.getParameter("TERM2"); //User's term2 input
inputField1 = request.getParameter("FIELD1"); //User's field1 select
inputField2 = request.getParameter("FIELD2"); //User's field2 select
inputOperator = request.getParameter("operator"); //User's operator

/*********************************************************
 *Build the query string:
 *There are two conditions:
 *1. the input term 1 is empty or the input term2 is empty.
 *   In this case, we need only SCORE(10) for scoring.
 *
 *2. The input term1 and term2 are both not empty.
 *   In this case, we need SCORE(10)+SCORE(20) for scoring.
 *********************************************************/

if (inputTerm1.length()!=0 && inputTerm2.length()!=0)
{
   if ( "content".equalsIgnoreCase(inputField1) |
        "content".equalsIgnoreCase(inputField2))
   {
      qry.append("SELECT patent.patentNum, title, SCORE(10), SCORE(20) ");
      qry.append("FROM patent, claim ");
      qry.append("WHERE patent.patentNum = claim.patentNum ");
   }
   else
   {
        qry.append("SELECT patentNum, title, SCORE(10), SCORE(20) ");
        qry.append("FROM patent ");
        qry.append("WHERE 1 = 1 ");
   }

   //add contains condition for field1
   qry.append("AND contains(");
   qry.append(inputField1);
   qry.append(", '");
   qry.append(inputTerm1);
   qry.append("', 10)>0 ");
```

```
    //add contains condition for field2
    qry.append(inputOperator);
    qry.append(" contains(");
    qry.append(inputField2);
    qry.append(", '");
    qry.append(inputTerm2);
    qry.append("', 20)>0 ");

    //order by scores.
    qry.append("ORDER BY (SCORE(10)+SCORE(20)) desc, patent.patentNum");
}

else
{
    if ( "content".equalsIgnoreCase(inputField1) |
         "content".equalsIgnoreCase(inputField2))
    {
        qry.append("SELECT patent.patentNum, title, SCORE(10) ");
        qry.append("FROM patent, claim ");
        qry.append("WHERE patent.patentNum = claim.patentNum ");
    }
    else
    {
         qry.append("SELECT patentNum, title, SCORE(10) FROM patent ");
         qry.append("WHERE 1 = 1 ");
    }

    //add contains condition for field1
    if (inputTerm1.length()!=0)
    {
        qry.append("AND contains(");
        qry.append(inputField1);
        qry.append(", '");
        qry.append(inputTerm1);
        qry.append("', 10)>0 ");
    }

    //add contains condition for field2
    if (inputTerm2.length()!=0)
    {
        qry.append("AND contains(");
        qry.append(inputField2);
        qry.append(", '");
        qry.append(inputTerm2);
        qry.append("', 10)>0 ");
    }
```

```
      //order by scores.
      qry.append("ORDER BY SCORE(10) desc, patent.patentNum");
    }

    /* Execute query */
    stmt = conn.createStatement();
    rs = stmt.executeQuery(qry.toString());

    /* List Table Names in HTML page */
%>

<HTML>
<BODY>

    <h1>e-Patent Project Database Query Result:</h1>
    <p align="left"><b>The retrieved information is based on your input
                  <font color="red"><%=inputTerm1%> </font> in <%=inputField1%> &
                  <font color="red"><%=inputTerm2%> </font> in <%=inputField2%>
                </b>
    <br>
    </p>

    <b>Please click on the patentNum to view the detailed information
      about your selected patent.</b>

    <table border="1" align="left">
       <tr>
          <th>
             Rank
          </th>
          <th>
             Score
          </th>
          <th>
             Patent Number
          </th>
          <th>
             Title
          </th>
      </tr>
      <% int i=0;
        while (rs.next())
        {
           i++;
      %>
```

```
      <tr>
        <td>
          <%=i%>
        <td>
          <font color="red">
           <% if (inputTerm1.length()!=0 && inputTerm2.length()!=0)
             {
          %>
                <%=Integer.parseInt(rs.getString("SCORE(10)"))+
                      Integer.parseInt(rs.getString("SCORE(20)"))%>
          <% }
          %>

           <% if (inputTerm1.length()==0 | inputTerm2.length()==0)
             {
          %>
                <%=Integer.parseInt(rs.getString("SCORE(10)"))%>
          <% }
          %>
          </font>

        </td>
        <td>
          <a href="showdetail.jsp?patentnum=
          <%=rs.getString("patentNum") %>"><%=rs.getString("patentNum")%></a>
        </td>
        <td>
          <%=rs.getString("title")%>

        </td>
      </tr>
      <% }
      %>
    </table>


<%
   /* Close result set and database connection */
   rs.close();


   conn.close();
}

/*catch exceptions here*/
catch (Exception e)
```

```
{
   System.out.println("DBclose failed");
   System.out.println(e.toString());
}
%>
</Body>
</HTML>
```

Appendix B— JSP: Exact Search

```
<%
/*
patentNumber.jsp
Search(exact) on patent Number(s) from Patent Project Oracle 8i database
If multi-patentnumbers, they are seperated by space.
Call using http://..../patentNumber.jsp
*/
%>


<% // ---------- Begin JSP Directives ---------- %>
<%@ page import="javax.servlet.*" %>
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.sql.*" %>
<%@ page language="java" %>

<%
// Begin variable definitions
String[] patentNumber;
patentNumber=new String[100];
int numberOf=0;

Statement stmt = null;
StringBuffer qry = new StringBuffer(1024);


/* Make connection to the database */
/* Set username and password for the database - blank in this case */
String db="sils";
String user= "ppproject";
String passwd = "*******";


/* Declare Connection and Result Set variables */
Connection conn = null;
ResultSet rs = null;

try
{
   /* Check we can find the database driver */

   DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

```
    conn = DriverManager.getConnection
              ("jdbc:oracle:oci8:@" + db, user, passwd);

    /*Get the input patent number(s)*/
    String numberInput = request.getParameter("patentnum");
    StringTokenizer st=new StringTokenizer(numberInput);

    /*seperate patent numbers and store the patent numbers
      into an array. */
    while (st.hasMoreTokens())
    {
        patentNumber[numberOf++]=st.nextToken();
     }


    /* Build query string */
    qry.append("SELECT patentNum, title FROM patent ");
    qry.append("WHERE patentNum = ");
    qry.append(patentNumber[0]);

    for(int i=1; i<numberOf;i++)
    {

       qry.append("OR patentNum = ");
       qry.append(patentNumber[i]);

    }

    qry.append("ORDER BY patentNum");

    /* Execute query */
    stmt = conn.createStatement();
    rs = stmt.executeQuery(qry.toString());
    /* List Table Names in HTML page */

%>

<HTML>
<BODY>

    <h1>e-Patent Project Database Query Result:</h1>
    <p align="left"><b>The retrieved information is based on your input
                    Patent Number = <%=numberInput %>
                </b>
    <br> </p>
```

```
    <b>Please click on the patentNum to view the detailed information
      about your selected patent.</b>

    <table border="1" align="left">
       <tr>
          <th>
             Patent Number
          </th>
          <th>
             Title
          </th>
       </tr>
       <% while(rs.next())
          {
       %>
       <tr>
          <td>
             <a href="showdetail.jsp?patentnum=
             <%=rs.getString("patentNum") %>"><%=rs.getString("patentNum")%></a>
          </td>
          <td>
             <%=rs.getString("title")%>
          </td>
       </tr>
       <% }
       %>
    </table>


<%
    /* Close result set and database connection */
    rs.close();
    conn.close();
}

catch (Exception e)
{
    System.out.println("DBclose failed");
    System.out.println(e.toString());
}
%>

</Body></HTML>
```