Analysis and pattern detection on large amounts of annotated sport motion data using standard SQL

Janez Perš,
Stanislav Kovačič
Faculty of Electrical Engineering
University of Ljubljana
Tržaška 25, SI-1000 Ljubljana, Slovenia
{janez.pers}@fe.uni-lj.si

Goran Vučkovič
Faculty of Sports
University of Ljubljana
Gortanova 22, SI-1000 Ljubljana, Slovenia

Abstract

This paper proposes a inexpensive and flexible way to analyze large amounts of sport motion data, which are generated by automated motion tracking methods and complemented with manual annotations. A database, obtained by tracking and annotating over 100 squash plays was used. The goal was to find a way to automatically detect certain kinds of play, activities, certain predefined scenarios and to generate various statistics about these activities – without hard-coding them in the executable code. We found that SQL-enabled databases provide a flexible and scalable solution to this problem. The examples of actual SQL queries for sport analysis are presented in the paper along with short tutorial on particular aspects of SQL language, which were exploited in our solution.

1. Introduction

The development in automated people tracking technology in the last decade has resulted in applications where large amounts of data can be generated with significantly less manual work than ever before [11, 12, 13]. One of such examples is the computer vision based tracking on sport videos, where, if certain conditions are met, computer tracks athletes with very little user intervention [14]. Such data can be used in *performance analysis* [10], giving sport community the feedback on player and team performance.

Years ago, analysis of sport matches was almost entirely manual, and the ability to gather certain kinds of data (like motion trajectories) was severely limited [7]. Before the introduction of automatic methods into the sports video analysis, every piece of information had to be entered into the computer by hand [3], and therefore manual annotations generated relatively small amounts of data.

However, by introducing the computer vision based processing of videos, the amount of data may increase dramatically. Widely used video standards (PAL, NTSC) assume frame rates of 25 or 30 video frames per second, and computer vision based tracking methods usually process *each captured video frame* for the greatest reliability. Therefore, in case of standard PAL videos, motion data of players is available at intervals of 40 milliseconds, and users have the ability to provide their manual annotations at the same temporal resolution, if they wish to. Computer user interfaces with integrated video players provide automatic synchronization of videos to the user annotations and therefore enable users to enter the annotations with much less effort. Additionally, modern desktop personal computers can calculate derived parameters (such as player velocity, acceleration and path length) with neglible computational cost.

As a consequence, the use of video based analysis techniques, coupled with computer vision based automatic tracking results in large amounts of output data. The problem is not limited only to video analysis. Alternative means of obtaining player motion data (e.g. radio waves-based localization) may provide even higher position sampling rates.

In the case of our research, 22 squash matches were recorded, which resulted in 140 GB of compressed (Motion-JPEG) video data. After automated tracking, manual annotation and automatic calculation of the derived parameters (e.g. velocity, distance), which took only several man-months, the resulting 1 Gigabyte of data was entered into Microsoft Access database.

In the remainder of the paper, the motivation for approaching the problem by using SQL enabled database engine is described. Next, short discussion on related work in the field of performance analysis is provided. Since we use SQL queries in a slightly unconventional way, we continue with short tutorial on some aspects of SQL syntax, used in majority of our queries. Finally, we present an example of actual source data and a few of actual SQL queries used to detect certain sequences of player activities and the corresponding statistics.

2 Motivation

Extracting relevant information from such large amount of data is a challenging task. For simple analysis, various spreadsheet packages (e.g. Microsoft Excel) may be used. However, when there is a large number of samples (matches, plays), such analysis may involve significant amount of manual work. Furthermore, users who are not familiar with advanced scripting capabilities of such packages will usually have no other option than to manually select all the required operations for each sample (e.g. one squash play) repeatedly, significantly increasing the possibility of human error and leaving behind only very coarse audit trail.

2.1 Why SQL?

For advanced analysis, spreadsheet tools may not be adequate at all. As an illustration, let us present a few problems that may appear during the course of research on squash play:

- Detection of simple activities under certain conditions, e.g. "How many times player A used stroke of type X in the particular region of the court? "
- Detection of complex scenarios, e.g. "How many times player A used stroke of type X, as a response to stroke of type Y by player B, and finished the point in the part of the court Z? "
- Analysis of other measurements e.g. "What were the players measured parameters (e.g. velocity), when performing using certain type of stroke or executing certain scenario? "

In our case, the types of player strokes and other activities have been entered as manual annotations and the rest of the data has been calculated from the tracking results. It is obvious that these problems could be solved easily by hard-coding appropriate detection algorithms in any of the widely used general-purpose programming languages. However, such a solution would have some significant drawbacks:

- Any addition of the new functionality (such as new type of activity, scenario or statistics) would require intervention of original developer, which can be costly and time consuming.
- Alternatively, a protocol would have to be devised for users to define new functionality, along with the appropriate user interface. This can be both costly to develop, time consuming in terms of user training, and can ultimately lead to similar limitations as described above, this time in the protocol or user interface.

The above limitations are particularly severe when such a tool is used in sports-related research, where researchers tend to define, validate and confirm or reject numerous hypotheses. Inclusion of program developers (essentially, general-purpose programmers) in this iterative loop along with numerous code modifications may be prohibitely expensive and impractical.

To overcome these problems, we decided to use a database engine with SQL interface capabilities. This way, any analysis of data is represented in a form of SQL query, and application for sport data analysis could consist simply of user friendly graphical user interface (GUI) and a set of SQL queries, which could be easily configured at runtime, without a need for original developer. In our research, we eventually scrapped the idea of an application altogether, due to availability of user-friendly interfaces for SQL engines (for example Microsoft Access).

3 Related work

The problem, described in this paper is distantly related to the large and well developed field of *data mining*, with one significant difference. Data mining aims to discover *implicit*, *previously unknown*, and potentially useful information from data [8], whereas we tried to detect and analyze well defined scenarios and player activities using standardized database interface language (SQL), and leave the process of actual knowledge discovery to scientists of the particular field - in our case, sport scientists.

Early researchers in the field of performance analysis relied heavily relying on optical methods of data comparison, such as transparencies [7]. To derive additional information from the manually entered data, (such as the length of intervals between certain annotations), custom applications were written for this task [3, 9].

In last few years, more generally oriented commercial packages gained popularity in the field of sport performance analysis. Authors used for example Theme software package, developed by Pattern Vision, which can be used for temporal pattern analysis [5]. Another popular tool which is used in performance analysis is Noldus Observer Pro [4].

4 Brief SQL tutorial

In this section we will present some very specific uses of SQL language. The examples, presented here, were tested in Microsoft Access 2003, which is a commercial product and can be obtained as part of the Microsoft Office 2003 suite of tools. However, many free (and much more powerful) SQL database engines are are also available today, such as MySQL or PostgreSQL. The examples throughout this paper would need only minor modifications to be used with other products.

We will not explain the process of entering SQL queries into the software – for details regarding the process of entering data and the use of SQL queries in particular product, reader should refer to the product documentation or one of many internet sources [1, 2]. Readers who would like smoother introduction into SQL language or have further

interest in SQL are advised to consult one of many introductory books on this topic, such as [6] or [15].

4.1 Database structure

SQL databases consist of one or multiple tables, which have a few columns and (usually) large number of rows. The columns and tables are named, while rows are not. Sometimes each row is assigned unique number (a key), which is helpful when a need to address individual rows arises.

A typical table used in sport performance analysis would have a column representing time of the measurement, two columns with X and Y positions per each player and one or more columns for different types of annotations. It would have as many rows as there are consecutive measurements in the particular data set.

To illustrate the results of SQL queries presented later in the text, we will start with a simplified table named *Raw*, with the contents as shown in Table 1.

Table 1. The contents of the table *Raw*

Seconds	Velocity	Region	Event
0	1.1	R01	A
1	1.5	R01	
2	1.2	R01	A
3	0.8	R02	
4	0.6	R01	B
5	0.5	R02	A
6	0.2	R02	B

This particular table contains the time of each measurement, velocity of a single player, the label of the region of the court where the player was present during the measurement (for which we assume that they have been derived from player positions in advance) and the manual annotations, which are of only two types: event A or event B. Some of the rows do not contain manual annotations, which is frequently the case when tracking methods with high temporal resolution are used (e.g. computer vision at 25 frames per second).

4.2 Basic analysis

The key to any analysis using SQL language is the `SELECT` statement. The structure of basic `SELECT` statements may look very similar to natural language description of a problem. The reason for this is the *non-procedural* nature of SQL language – the queries tell SQL *what* the user needs, not *how* to obtain it. Let's start with a simple listing:

Problem 4.1 *Show all columns except velocity, for all the measurements where the player was in region R01.*

```
SELECT Seconds, Region, Event FROM Raw
WHERE Region='R01';
```

The structure of this simple SQL query is as follows: the list, immediately following the `SELECT` keyword specifies which columns need to be extracted from the source table. The name of the data source (table *Raw*) immediately follows the `FROM` keyword, and the text following the `WHERE` keyword describes the conditions that individual rows have to fulfill to be displayed. Most of the SQL queries in this paper will have those three basic elements. In addition to displaying data from the tables, SQL can be used to do more powerful analysis:

Problem 4.2 *Count all the events of type B in the table Raw.*

```
SELECT COUNT(*) FROM Raw WHERE
Event='B';
```

Problem 4.3 *Calculate the average velocity for all the measurements where the event of type A was annotated and player was in the region R01.*

```
SELECT AVG(Velocity) FROM Raw WHERE
(Region='R01') and (Event='A');
```

The last example is based on the function `AVG`, which calculates the average value of a particular column (which was previously filtered through the conditions following the `WHERE` keyword, of course).

4.3 Purging data

As illustrated in Table 1, manually annotated data often does not contain annotations in every row of the table. As an obvious consequence, two annotations can be separated by an arbitrary number of empty cells, which makes pattern detection in SQL impossible. To solve this problem, tables can be preprocessed using various criteria, to remove rows which are not needed for the particular kind of analysis.

Let us create new table, named *Processed*, which will contain purged data. For this purpose a new type of command is used:

```
CREATE TABLE Processed (ID
AutoIncrement, Seconds float, Velocity
Float, Region text, Event text);
```

The table is initialized to hold all the columns of the table *Raw*, plus the new column named *ID*. This column will serve as a tool for relative addressing of the rows in the pattern detection step. It is sufficient to know that this special column, declared as *AutoIncrement* contains integer values which are incremented by one at each row in the table.

Let us fill this new table, *Processed*, with the data from the table *Raw*, filtering out all the rows that do not contain any value in the *Event* column:

```
INSERT INTO Processed ( Seconds,
Velocity, Region, Event ) SELECT
Seconds, Velocity, Region, Event FROM
Raw WHERE Event IS NOT NULL;
```

The contents of the table *Processed* after the execution of this command is shown in Table 2.

Table 2. The contents of the table *Processed*

ID	Seconds	Velocity	Region	Event
1	0	1.1	R01	A
2	2	1.2	R01	A
3	4	0.6	R01	B
4	5	0.5	R02	A
5	6	0.2	R02	B

It can be seen that the second table now contains only those rows from the table *Raw* that had non-empty *Event* cells. It can be also seen that the *Seconds* column now contains non-contiguous blocks of numbers, while the *ID* column contains linearly increasing integer values.

4.4 Detecting patterns

After the purging step, the *Processed* table can be analyzed further to reveal any known temporal patterns in any of the columns. Since we purged the original data with the intent to discard empty cells in the *Event* column, we may proceed with detection of the particular sequence of events.

For this purpose, advanced form of `SELECT` query can be used.

```
SELECT t1.ID, t1.Seconds, t1.Event
FROM Processed AS t1 INNER JOIN
Processed AS t2 ON t1.ID+1=t2.ID WHERE
(t1.Event='A') and (t2.Event='B');
```

The query contains `INNER JOIN` commands and table aliases, which we will not describe in detail. The query is made up of the following parts:

- Two `AS` keywords, specifying that the single table *Processed* should be assigned two aliases, *t1* and *t2*. As a consequence, individual columns can be addressed by referring to either of those two aliases and a column name, separated by a dot.
- The `ON` statement, where the temporal constraint is defined. In our case, it specifies that for each row processed, the tables *t1* and *t2* should have ID numbers for exactly value of 1 apart. This in fact causes the database to compare two consecutive rows in the table, but could be generalized to specify different kind of constraints, including larger intervals between rows.
- Slightly more complicated `WHERE` part, where the condition for the contents of the two rows being compared is specified. In our case, it specifies that the table *t1* should contain the annotation A in its *Event* field, and that the table *t2* should have the annotation B in its *Event* field. Together with the `ON` statement that effectively means that the query looks for the temporal sequence A,B through the whole table. Again, this condition can be generalized as well.
- Slightly expanded `SELECT` part, where the columns which should be printed when all the conditions are met are named.

It is possible to link together more conditions, for example to detect longer temporal patterns in data. This can be accomplished by nesting several `INNER JOIN` statements.

Such type of query can be easily adapted to perform more advanced analysis on the detected patterns. For example:

Problem 4.4 *Calculate the average value of velocities at the points where the pattern A,B in the was detected in the* Event *column.*

```
SELECT AVG(t1.Velocity) FROM Processed
AS t1 INNER JOIN Processed AS t2 ON
t1.ID+1=t2.ID WHERE (t1.Event='A') and
(t2.Event='B');
```

The query above calculates the average over the velocities for each row where event A was present, but only those rows when it was present as a start of sequence A,B are included in the calculation.

It can be seen that many different detectors can be built using different combinations of data purging and `INNER JOIN` statements. With properly formatted input data it would be possible to do the following:

- Detect and count the transitions between different regions of the court. Analyze other player parameters (e.g. velocity, direction) when such transition is detected - in the same query.
- Detect and count the transitions between different velocity classes. Detect interesting patterns in motion (e.g. changes of direction).
- Measure intervals between the specified activities.
- Make complex queries which would detect and observe patterns which span multiple columns.

5 SQL in squash match analysis

The approach described in previous section has been extensively used to process numeric data from 22 squash matches. First, digitized videos have been processed offline in two passes by the operator-supervised computer vision based tracking application. In the first pass, operator manually selected both players, started the automatic tracker and supervised the tracking process. Interventions were rare and the tracker easily processed up to 20 frames per second. In this pass, the software recorded the time and position of both players. In the second pass, operator manually annotated the video using a set of buttons on the application's user interface. He switched the play mode between passive phase and rallies and noted each stroke of any of the players by clicking on the appropriate button.

When the stroke was recorded, the program stopped and allowed the operator to precisely click on the ball at the moment when hit by a racket. Then operator selected approximate height of the ball and chose two additional event descriptions. In this phase, the basic information about the recorded event (annotation), racket position and additional info was recorded. After completing the tracking and annotation process, motion data was smoothed, and finally, several derived parameters were calculated both from of player motion data and annotation data. Data for each squash play was exported to tab separated file and loaded into the Microsoft Access database. Each play was represented as one table in the database.

5.1 Data format

Table 3 shows short excerpt from one of the tables. To preserve clarity, only motion data for the first player are shown. The eight rightmost columns are reserved for annotation data, some of which was derived automatically (region of the court, for example). Table 4 explains the contents of the individual columns in the each table. The symbol TableName in queries denotes the name of the table for which the statistics is calculated.

Table 4. Column contents legend.

Column	Meaning
Frames	Number of video frames from the play start
Seconds	Elapsed time from the play start [s]
X	X player coordinate [m]
Y	Y player coordinate [m]
V	Player velocity [m/s]
Acc	Player acceleration $[m/s^2]$
Dist	Cumulative path length [m]
DiffDist	Differential of path length[m]
Rally	Rally or passive phase [R or P]
Region	Region of the court where the player was
Event	Major annotation mark - type of the stroke
Descr1	2nd annotation mark (let, stroke, error...)
Descr2	3rd annotation mark (forehand, backhand)
Event X	X stroke coordinate [m]
Event Y	Y stroke coordinate [m]
EventRegion	Stroke region of the court

5.2 Sample SQL queries

In this section, we will present some of the problems that have been solved constructing the appropriate SQL query on the data, presented in Table 3.

Problem 5.1 *What was the cumulative duration of rallies and passive phases of the play (at 25 frames/measurements per second)?*

```
SELECT Rally, COUNT(*)/25 AS
RallySeconds FROM TableName GROUP BY
Rally;
```

Problem 5.2 *How many rallies and passive phases are there in particular play?*

```
SELECT DISTINCT t2.Rally, Count(*)
AS NumPhases FROM TableName AS
t1 INNER JOIN TableName AS t2
ON t1.Frames+1=t2.Frames WHERE
((t1.Rally='R') AND (t2.Rally='P'))
OR ((t1.Rally='P') AND (t2.Rally='R'))
GROUP BY t2.Rally;
```

Note - we are actually counting changes between rallies and passive phases in this query, that's why `INNER JOIN` is needed.

Problem 5.3 *What is the average velocity of a player during the rallies and during the passive phases?*

```
SELECT Rally, Avg(V) AS AverageVelocity
FROM TableName GROUP BY Rally ;
```

Problem 5.4 *Generate a table of all rallies, calculate their durations, calculate player path lengths in each of the rallies and calculate average velocities for each rally.*

This problem requires purging of the unnecessary data and creation of a temporary table named TableRallies:

```
INSERT INTO TableRallies (Rally,
Seconds, Dist) SELECT DISTINCT t2.Rally
AS Rally, t2.Seconds AS Seconds,
t2.Dist AS Dist, FROM TableName
AS t1 INNER JOIN TableName AS t2
ON t1.Frames+1=t2.Frames WHERE
((t1.Rally='R') AND (t2.Rally='P'))
OR ((t1.Rally='P') AND (t2.Rally='R'));
```

In the second step, statistics can be generated from the intermediate table named TableRallies, which includes integer key named ID:

```
SELECT DISTINCT t1.Rally,
(t2.Seconds-t1.Seconds) AS
RallyDuration, (t2.Dist-t1.Dist)
AS PathLength, (t2.Dist-t1.Dist)
/ (t2.Seconds-t1.Seconds) AS
AverageVelocity FROM TableRallies AS
t1 INNER JOIN TableRallies AS t2 ON
t1.ID+1=t2.ID WHERE (t1.Rally='R');
```

6 Conclusion

We have presented a flexible way to analyze annotated motion data without the need for writing custom applications in general purpose languages. There is no doubt that

Table 3. Excerpt from a source data - raw output from tracking and annotating application.

Frames	Seconds	X	Y	V	VClass	Acc	Dist	DiffDist	Rally	Region	Event	Descr1	Descr2	EventX	EventY	EventRegion
2750	110	7.38	2.11	0.84	walk	-5.3	125.13	0.0336	R	R22				0	0	
2751	110.04	7.38	2.09	0.62	walk	-5.54	125.15	0.0247	R	R22				0	0	
2752	110.08	7.37	2.07	0.4	walk	-5.37	125.17	0.0161	R	R22				0	0	
2753	110.12	7.37	2.06	0.23	walk	-4.26	125.18	0.0093	R	R22	LL1	R	b	7.36	1.07	R29
2754	110.16	7.37	2.06	0.17	walk	-1.69	125.19	0.0066	R	R22				0	0	
2755	110.2	7.38	2.05	0.26	walk	2.24	125.2	0.0102	R	R22				0	0	
2756	110.24	7.39	2.04	0.32	walk	1.71	125.21	0.0129	R	R22				0	0	
2757	110.28	7.4	2.04	0.26	walk	-1.61	125.22	0.0104	R	R22				0	0	
2758	110.32	7.41	2.05	0.23	walk	-0.65	125.23	0.0093	R	R22				0	0	
2759	110.36	7.4	2.07	0.56	walk	8.05	125.25	0.0222	R	R22				0	0	
2760	110.4	7.4	2.11	0.81	walk	6.38	125.28	0.0324	R	R22				0	0	
2761	110.44	7.41	2.14	0.87	walk	1.53	125.32	0.0349	R	R22				0	0	
2762	110.48	7.44	2.17	0.95	walk	1.97	125.36	0.038	R	R22				0	0	
2763	110.52	7.47	2.19	1.01	walk	1.59	125.4	0.0406	R	R22				0	0	
2764	110.56	7.5	2.21	0.84	walk	-4.29	125.43	0.0337	R	R22				0	0	
2765	110.6	7.51	2.23	0.71	walk	-3.29	125.46	0.0284	R	R22				0	0	
2766	110.64	7.51	2.27	0.84	walk	3.21	125.49	0.0336	R	R22				0	0	
2767	110.68	7.53	2.3	0.96	walk	3.1	125.53	0.0385	R	R22				0	0	
2768	110.72	7.57	2.33	1.26	walk	7.46	125.58	0.0505	R	R22				0	0	
2769	110.76	7.63	2.35	1.6	jogging	8.5	125.65	0.0641	R	R22				0	0	
2770	110.8	7.69	2.37	1.59	jogging	-0.18	125.71	0.0638	R	R22				0	0	
2771	110.84	7.73	2.4	1.26	walk	-8.42	125.76	0.0503	R	R22				0	0	
2772	110.88	7.75	2.44	1.05	walk	-5.09	125.8	0.0422	R	R22				0	0	
2773	110.92	7.75	2.49	1.23	walk	4.32	125.85	0.0491	R	R22				0	0	
2774	110.96	7.73	2.54	1.41	jogging	4.54	125.91	0.0564	R	R22				0	0	
2775	111	7.7	2.59	1.44	jogging	0.73	125.96	0.0575	R	R22				0	0	

such task could be achieved by hard-coding the appropriate queries in the general purpose language, however, besides flexibility, the use of SQL gives this approach platform independence and scalability. We are convinced that such approach will make even more sense in a future in various applications beyond the sport domain (e.g. surveillance), when tracking algorithms will be even more powerful and will produce a massive flow of motion measurements and perhaps automated annotations. The natural choice for storing such data will be large databases, and natural interface for various kind of motion and behavior analysis would be the queries written in SQL language.

Limitations of SQL language are reflected in limitations of such approach as well. One of the weaknesses of standard SQL is the inter-row processing, which requires one `INNER JOIN` per each inter-row comparison. The queries get increasingly complicated and time consuming as the number of conditions in definition of pattern increases. Nevertheless, we extensively used the Microsoft Access database this way, and more powerful database engines may offer additional non-standard extensions which may resolve those limitations.

Acknowledgement

Authors wish to thank Marko Pavlišič for helpful hints regarding advanced features of SQL language.

References

[1] Access 2000 tutorial. `http://www.fgcu.edu/support/office2000/access/`.

[2] Mysql tutorial. `http://dev.mysql.com/doc/mysql/en/tutorial.html`.

[3] A. Ali and M. Farrally. A computer-video aided time motion analysis technique for match analysis. *The Journal of Sports Medicine and Physical Fitness*, 31(1):82–88, March 1991.

[4] D. Bishop. Performance analysis: What is performance analysis, and how can it be integrated within the coaching process to benefit performance? `http://www.pponline.co.uk/encyc/performance-analysis.html`.

[5] A. Borrie, G. K. Jonsson, and M. S. Magnusson. Temporal pattern analysis and its applicability in sport: an explanation and exemplar data. *Journal of Sports Sciences*, 20(10):845–852, 2002.

[6] D. Dicken and K. Thompson. *Learn SQL in a weekend*. Premier Press, 2002.

[7] W. S. Erdmann. Gathering of kinematic data of sport event by televising the whole pitch and track. In R. Rodano, editor, *Proceedings of 10^{th} ISBS symposium*, pages 159–162. International Society of Biomechanics in Sports, 1992.

[8] W. Frawley, G. Piatetsky-Shapiro, and C. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, pages 57–70, Fall 1992.

[9] M. Hughes, F. I. M., and N. P. A video-system for the quantitative motion analysis of athletes in competitive sport. *Journal of Human Movement Studies*, 17:217–221, 1989.

[10] M. D. Hughes and R. M. Bartlett. The use of performance indicators in performance analysis. *Journal of Sports Sciences*, 20(10):739–754, 2002.

[11] S. S. Intille and A. F. Bobick. Visual tracking using closed-worlds. In *Proceedings of the Fifth International Conference on Computer Vision ICCV '95*, pages 672–678, MIT, Cambridge, MA, June 20-23 1995.

[12] C. J. Needham and R. D. Boyle. Tracking multiple sports players through occlusion, congestion and scale. In *12th British Machine Vision Conference, BMVC01*, pages 93–102, Manchester, UK, September 2001.

[13] J. Perš, M. Bon, S. Kovačič, M. Šibila, and B. Dežman. Observation and analysis of large-scale human motion. *Human Movement Science*, 21:295–311, 2002.

[14] J. Perš, G. Vučkovič, S. Kovačič, and B. Dežman. A low-cost real-time tracker of live sport events. *ISPA 2001: Proceedings of the 2nd international symposium on image and signal processing and analysis in conjunction with 23nd int'l conference on information technology interfaces, Pula*, pages 362–365, June 19-21,2003.

[15] R. Plew and R. Stephens. *Teach Yourself SQL in 21 Days*. Sams, 2002.