A Relational Database for Human Motion Data

Qaiser Riaz, Björn Krüger, and Andreas Weber

Insitute for Computer Science II, Bonn University, Germany

Abstract. Motion capture data have been widely used in applications ranging from video games and animations to simulations and virtual environments. Moreover, all data-driven approaches for analysis and synthesis of motions are depending on motion capture data. Although multiple large motion capture data sets are freely available for research, there is no system which can provide a centralized access to all of them in an organized manner. In this paper we show that using a relational database management system (RDBMS) to store data does not only provide such a centralized access to the data, but also allows to include other sensor modalities (e.g. accelerometer data) and various semantic annotations. We present two applications for our system: A motion capture player where motions sequences can be retrieved from large datasets using SQL queries and the automatic construction of statistical models which can further be used for complex motion analysis and motions synthesis tasks.

1 Introduction

Motion Capturing (mocap) has become a standard technique in the last decades. Various data-driven applications like capturing with low cost sensors (e.g. Kinect), action recognition and motion synthesis have been developed. All these data-driven methods require high quality motion capture data.

For research purposes multiple datasets [1–4] containing tens of gigabytes of mocap data are published and available for free download. These so called *databases* are usually available as collections of files in different formats, like C3D, BVH and ASF/AMC. Currently, the two largest, well established, and freely available collections of mocap data are the CMU [1] and HDM05 [2] databases where the data is organized in flat files. The file names do not include any information about the nature of the event stored in the file. An indication of the content of each file is given as rough textual description only.

For the CMU data collection, textual descriptions are given for each motion file on the web page. In addition, links to motion files are sorted by subjects and categories, such as *Human Interaction*, *Locomotion*, *Physical Activities & Sports*. Each file name contains the subject number and an event number only. For the HDM05 data collection, a technical report is available, where a script with the description of the tasks the subjects had performed is published. Here, the file-name refers to the subject, a chapter in this script and thus, gives an indirect indication on the actual content. Additionally a frame rate is stored in the HDM05 file names.

On the one hand there exists no centralized policy for frame accurate annotations of motion capture data. Annotations on the frame level are only available for few example motion sequences, as shown for comparison in the context of motion segmentation [5, 6]. This makes it difficult to search the available data for contents of interest. It is not directly possible to extract a sub part of a relevant information e.g. extracting those frames where subject is in T-pose. To this end, whole files have to be parsed and relevant contents have to be extracted by hand. Another disadvantage of flat files is an inability to query, search, and retrieve information in a structured manner. On the other hand a huge amount of data-driven techniques were developed (see [7–10] for some examples) that make use of carefully selected motion data. Our goal is to provide tools that simplify and speed-up the work flow of data selection and model building for such data-driven approaches.

The main contributions of this paper are:

- We present a flexible Entity-Relationship (ER) model that is capable to handle motion capture data from various public available datasets.
- We come up with solutions to incorporate additional sensor data and data that can be derived from the original measurements.
- We show that hierarchical annotations can be handled to describe the content of the motion data.
- We show that default applications can be supported by relational databases.

The remainder of this work is organized as follows: We give an overview on previous and related work in Section 2. The ER model of our database system is explained in detail in Section 3, while we show some basic operations that are available in Section 4. In Section 5 we consider details of the performance optimization of the database scheme. Exemplary applications are shown in Section 6 and the work is concluded in Section 7.

2 Related Work

To handle the growing amount of mocap data, many alternative methods for fast search and retrieval have been proposed by the research community. Based on frame by frame comparison, *Match Webs* [11] were introduced to come up with a efficient index structure. Due to the quadratic complexity it is not possible to build index structures on all currently available mocap data with this method. To avoid quadratic complexity of frame by frame comparisons, segmentation based methods were developed [12]. Cluster based methods classify motion database into small clusters or groups. The classification is either based on similar frames in different motions [13] or on clustering similar poses [14]. Binary geometric features [15] can be used to tackle this kind of segmentation problems. While methods based on boolean features can not describe close numerical similarity, this is the case when low dimensional representations of the motion data are used for comparison. To compute low-dimensional representations of mocap data principal component analysis (PCA) is a well established method [16, 17].

Another way to come up with low dimensional representations is to compute feature sets in a normalized pose space [18]. Dimensionality reduction is achieved by using a subset of joint positions only. For the authors, a feature set consisting of hand, feet and the head positions is the one of choice. They also show, that it is possible to search these feature sets efficiently by employing a kd-tree as index structure.

This retrieval technique is adopted for the database architecture proposed by [19]. In this work the authors focus on both, data compression and retrieval of motion capture data. A compression ratio of 25% of the original database size is achieved with their method. To this end a curve simplification algorithm is applied to reduce the number of frames to 20% of the original ones. An adaptive k-means clustering algorithm is used to group similar motion clips together. In addition, a three-step motion retrieval method is used which accelerates the retrieval process by filtering irrelevant motions. A database architecture consisting of data representation, data retrieval and selection modules has been proposed [20]. An in-memory internal representation of the motion data consisting of a collection of poses is created via a data representation module. The data retrieval and selection module queries this in-memory representation using PhaseQuery and results are expressed as sequences of segments. The search and retrieval time is almost real-time for small data sets but it increases dramatically for larger data sets. The in-memory condition requires enough physical memory to load large data sets.

For annotation purposes, semi-supervised techniques were employed. Based on a support vector machine (SVM), a database of football motions was annotated [17]. Such kind of annotations are transferred from motion capture data to video sequences in [21]. To visualize and explore huge data sets hierarchical clustering on normalized pose features [22] was used to obtain an overview on huge data collections. In opposite, Lin [23] presents a system where Kinect queries are used to search and explore mocap datasets that are modeled as sets of 3-attribute strings.

3 Database Architecture

The Entity-Relationship (ER) Model of the proposed database architecture is shown in Figure 1. The core schema is divided into four different categories:

1. **Controller Entity:** The heart of the proposed schema, which controls the flow of information.
2. **Sensor-specific Entities:** To handle sensor-specific data for each sensor.
3. **Annotations Control Entities:** Control annotation policy mechanism.
4. **Derived Entities:** To entertain non-mocap data which is computed from the mocap data.

We will briefly explain each of these categories in the following subsections.

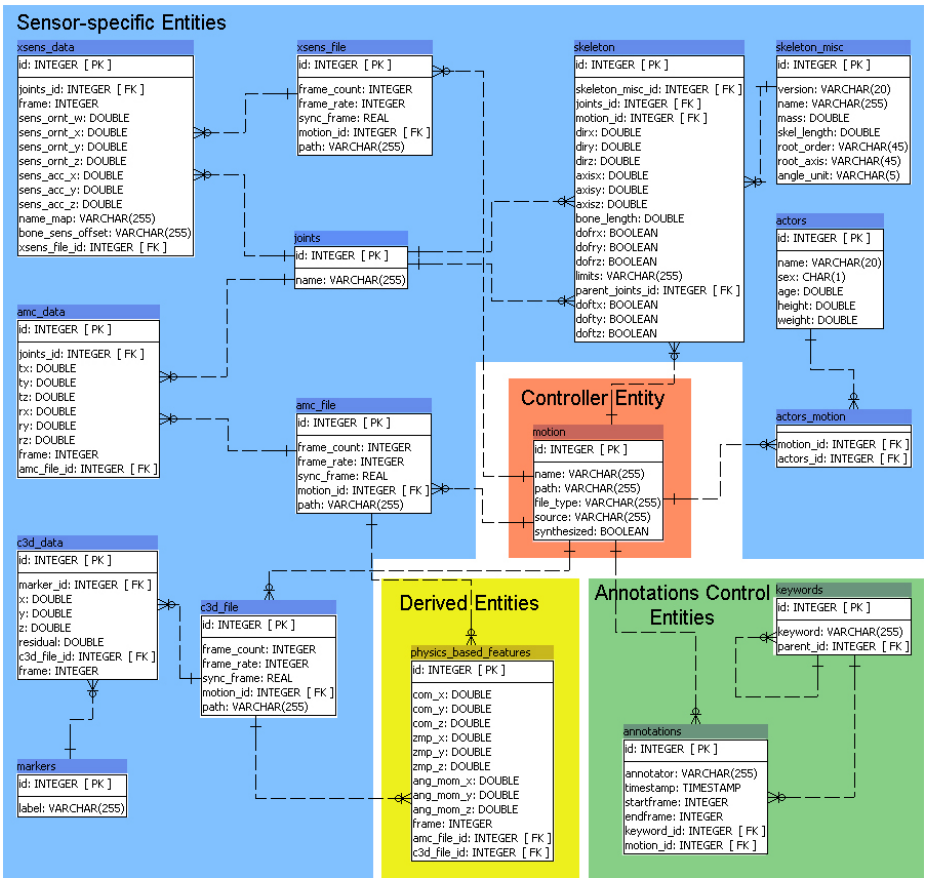

Fig. 1. Entity Relationship (ER) model of the proposed database architecture. The core schema is divided into four different categories, each of which handles an aspect of the proposed schema.

3.1 Controller Entity

The purpose of the controller entity is to control the flow of information in a logical manner. It functions as the heart of the schema providing a logical bounding among different entities. All vital information passes through the controller entity. In our proposed database architecture, the *motion* entity acts as the controller entity (Figure 1). The controller entity has a simple schema with certain general attributes of a file such as 1) *name:* stores actual name of the file, 2) *file_type:* stores type of the file e.g. amc, c3d, 3) *source:* stores the source of the file e.g. HDM05, CMU, and 4) *synthesized:* a boolean flag to indicate if the motion data is an actual recording or an artificial motion obtained by some motion synthesis procedure.

Table 1. Sensor-specific Entities, their attributes and description of each attribute.

Entities	Attributes	Description
amc_file, c3d_file, xsens_file	frame_count, frame_rate	Total frames and frame rate
	sync_frame	Synchronization frame
	path	Physical path on HDD
amc_data	tx, ty, tz	Translation (x, y, z)
	rx, ry, rz	Rotation (x, y, z)
	frame	Frame number
c3d_data	x, y, z	3D coordinates
	residual	Residual
	frame	Frame number
xsens_data	sens_ornt_w, sens_ornt_x, sens_ornt_y, sens_ornt_z	Sensor orientation (w, x, y, z)
	sens_acc_x, sens_acc_y, sens_acc_z	Sensor acceleration (x, y, z)
	name_map	Name map
	bone_sens_offset	Bone sensor offset
	frame	Frame number
skeleton	dirx, diry, dirz	Direction (x, y, z)
	axisx, axisy, axisz	Axis (x, y, z)
	dofrx, dofry, dofrz	Degree of freedom - rotation (x, y, z)
	doftx, dofty, doftz	Degree of freedom - translation (x, y, z)
	bone_length, limits	Bone length, limits
skeleton_misc	version, name, mass	Sensor's version, name, and mass
	skel_length	Skeleton length
	root_order, root_axis	Order (rotation, translation) and Axis (x,y,z) of the root
	angle_unit	Unit of the angle

3.2 Sensor-specific Entities

Most of the entities in our proposed database schema are sensor-specific. Sensor-specific entities, as the name indicates, are used to store sensor specific information in the database. In order to achieve a flexible design of the database schema, general properties of each recording are stored in separate entities (name of the entity in a format: *sensor name + an underscore + 'file'* e.g. *c3d_file*) and actual data is stored in separate entities (name of the entity in a format: *sensor name + an underscore + 'data'* e.g. *c3d_data*). Each sensor can have any additional supporting entities. For example to store AMC data, the general properties are stored into *amc_file* table and the actual data is stored into *amc_data* table. The supporting entity in this case is *joints* table. We have processed and stored data from different sensors in our proposed database, which include:

ASF Data: The ASF skeleton data is distributed into two entities namely *skeleton_misc* and *skeleton*. The *skeleton_misc* entity stores the general attributes

of skeleton while the *skeleton* entity stores specific skeleton data of each joint in each frame. The attributes of both entities are described in Table 1.

AMC Data: The AMC motion data is stored into two mandatory entities *amc_file* and *amc_data* and a supporting entity *joints*. The *amc_file* entity stores general information about the data such as *frame count*, *frame rate*, *synchronization frame* etc. A synchronization frame is used to overcome synchronization problem amongst different sensor systems, which occurs when a single motion is simultaneously recorded by multiple motion capture devices. The *amc_data* stores rotation and translation data for each joint in each frame. The *joints* entity has a *one-to-many* relationship with the *amc_data* and provides an easy mechanism of joint-based data search using standard SQL statements. The attributes of AMC data entities are described in Table 1.

C3D Data: The C3D data is stored into two mandatory entities *c3d_file* and *c3d_data* and a supporting entity *markers*. Like the *amc_file*, the *c3d_file* entity also stores general information about the data such as *frame count*, *frame rate*, *synchronization frame* etc. The *c3d_data* entity stores 3D information of each marker in each frame. The *markers* entity has a *one-to-many* relationship with the *c3d_data*. The database can be queried based on markers to fetch data of a particular marker using standard SQL statements. The attributes of C3D data entities are explained in Table 1.

Accelerometer Data (Xsens): The accelerometer data is not available in CMU or HDM05 mocap data sets. However, some recordings of accelerometer data were captured later on and we have included these data sets in our database schema. This shows the flexibility and extensibility of our database architecture that any new sensor can be easily integrated within the existing schema. The data has been recorded using Xsens's MTi accelerometer [24]. In order to store data; two mandatory entities *xsens_file* and *xsens_data* and a supporting entity *joints* are used. The *xsens_file* has same attributes as *amc_file* and *c3d_file*. The *xsens_data* entity stores orientation and acceleration data of each joint in each frame. The *joints* entity has a *one-to-many* relationship with the *xsens_data*. The attributes of accelerometer data entities are explained in Table 1.

3.3 Annotations Control Entities

Annotations control entities are one of the important entities in the proposed database architecture. These entities define an annotation policy mechanism and provide an easy way to query the database based on an event keyword. In the proposed database architecture, two entities have been introduced to handle annotations. The *annotations* entity stores the general attributes of an annotation such as *start frame*, *end frame*, *timestamp* etc. the *keywords* entity serves as a dictionary of annotations and has a *one-to-many* relationship with the *annotations* entity. It also has a *self-relation* to maintain a *hierarchical relationship*

Table 2. Annotations Control Entities, their attributes and description of each attribute.

Entities	Attributes	Description
annotations	annotator	Name of the annotator
	timestamp	Record creation timestamp
	startframe	Starting frame number of the motion
	endframe	Ending frame number of the motion
keywords	keyword	Keyword
	parent_id	A self relation, *null* if no parents

Table 3. Derived entities - physics based features. Physics based features are derived from mocap data sets and do not exit on their own.

Entity	Attributes	Description
physics based features	com_x, com_y, com_z	Center of Mass (x, y, z)
	zmp_x, zmp_y, zmp_z	Zero Moment Point (x, y, z)
	ang_mom_x, ang_mom_y, ang_mom_z	Angular Momentum (x, y, z)
	frame	Frame number

between different keywords. The *hierarchical relationships* are parent- child relationships and define annotations from high-level general terms to low-level more specific terms. For example, in HDM05, a *'jumping jack'* motion is a child of the *'workout'* event and a grand child of the *'sports'* event. So the *parent_id* of the *'jumping jack'* will be the *id* of the *'workout'* and the *parent_id* of the *'workout'* will be the *id* of the *'sports'*. The attributes of annotations control entities are expressed in Table 2.

3.4 Derived Entities

In the proposed database architecture, there are certain entities which are derived from the existing mocap data sets. These entities do not exist in any freely available mocap data set. However, they are required in many research activities and researchers have to manually compute them whenever required. A good example of derived entities is physics based features such as center of mass, zero moment point, and angular momentum, which can be computed through kinematics [25]. To entertain these features, a separate entity namely *physics_based_features* has been created. The attributes of the *physics_based_features* are explained in Table 3. This table has no real data at the moment and computing and dumping physics based features into the database will be carried out in near future.

4 Basic Database Operations

4.1 Processing and Storing Mocap Data into Database

We have used PostgreSQL, version 9.0 - 64 bit, to store extracted data from mocap data sets. PostgreSQL is a widely used object-relational database management system which is freely available under PostgreSQL license. A database

has been created using our proposed database schema. In order to extract data from different formats and store into database, special scripts are written in Visual C++ and Matlab. These scripts read each file of a particular format, extract data of interest, and generate text files with structured queries. These queries are then executed under PostgreSQL environment to store data into different tables. In order to optimize insertion process by minimizing data insertion time, concepts of bulk data insertion are used.

4.2 Retrieving Collections

Collections can be retrieved using standard SQL statements. In the upcoming subsections, we will give some examples of retrieving collections using standard SQL queries.

Retrieving All Actors: This is a simple example of retrieving data of all *actors*. Each event in mocap data is performed by one or more actors and motions can be retrieved based on actor information.

```
select * from actors;
```

Retrieving All Annotation Keywords: This is another simple example of retrieving all *keywords*. Each event in mocap data is annotated through a keyword, which explains the nature of the event.

```
select * from keywords;
```

Retrieving Annotations of a Specific Event Group: Sometimes one is interested to find all annotations of a specific group of events e.g. finding all '*sports*' annotations. In this example we show how one can retrieve annotations of a specific event group. The event group of interest, in this case, is '*sports*'.

```
select keyword from keywords
 where parent_id = (
  select id from keywords
   where keyword=''sports'')
```

Retrieving Motion Information of an Event: This example shows how to retrieve motion IDs of all motion records for *'dancing'* event. These IDs can be used in later steps to retrieve actual motion data.

```
select m.id from motion m,
 annotations a, amc_file af,
 keywords k
 where m.id=a.motion_id
  and a.keyword_id=k.id
  and af.motion_id=m.id
  and k.keyword=''dancing''
```

Retrieving Synchronized C3D Data: In this example, we show how to retrieve synchronized data based on *syn_frame* value. The synchronization frame, *syn_frame*, is used to overcome synchronization problem amongst different sensor systems, which occurs when a single motion is simultaneously recorded by multiple motion capture devices. The data-type of the *syn_frame* attribute is *real* and stores synchronization time in seconds. In the presence of this time, retrieving synchronized data is very easy and straight forward as shown in the following query.

```
select * from c3d_data
 where c3d_file_id=1 and frame >
   (select sync_frame*frame_rate
    from c3d_file where id=1)
```

Retrieving AMC Data: In this query we extract all AMC data for *'throwing'* event where actor is *'mm'* and source of mocap data is *'HDM05'*. This is a complex query as it involves multiple joins among various entities such as *actors*, *motion*, *amc_file* etc.

```
select * from amc_data
 where amc_file_id in
  (select af.id from amc_file af,
  motion mo where
  mo.source =''HDM05''
  and mo.id=af.motion_id
  and mo.id
   in (select m.id from motion m,
    annotations a,
    actors_motion am
    keywords k, actors ac
     where m.id=a.motion_id
      and a.keyword_id=k.id
      and ac.id=am.actors_id
      and am.motion_id=m.id
      and ac.name=''mm''
      and k.keyword=''throwing''))
```

5 Database Performance Evaluation

5.1 Performance Optimization

Before we evaluate the performance of the presented database scheme, we give some insights of the steps taken for optimization of the database structure.

The size of the database on hard disk is approximately 61 GB after parsing and inserting data from all ASF/AMC and C3D files for both *HDM05* and *CMU*. Entities *amc_data* and *c3d_data* are the largest entities having approximately *90 million* and *130 million* records respectively. Hence, an optimization policy is required in order to minimize database search and retrieval time and maximize the system's response time. Indexing is one of the widely used optimization techniques in relational database management systems. PostgreSQL provides several built-in indexing algorithms such as

Table 4. A comparison of performance optimization with and without indexing. The data search and retrieve time has substantially decreased by introducing binary tree based indexes.

Retrieving	Execution Time (ms)	
	Not Indexed	Indexed
All Actors	51	11
All Keywords	13	10
Annotations of an Event Group	12	11
Motion Information of an Event	111	30
Synchronized C3D Data	244,770	18,029
AMC Data	217,795	8,132

B-tree, Hash, GiST and GIN [26]. PostgreSQL uses B-tree as default indexing algorithm [26]. We have created indexes using B-trees on primary keys of both tables. We have also created indexes using B-trees on foreign keys to minimize search and retrieval time.

The trade-off of using indexes is slow data insertion as indexes are updated upon each insertion. However, mocap data sets are not frequently updated so one can compromise on slow data insertion over fast find and fetch. Alternatively, indexes can be dropped during insertion to speed up the insertion process and can be regenerated afterward. We have executed all queries listed in the section *Retrieving Collections* 4.2 with and without indexing and the results are presented in Table 4. The comparison clearly indicates substantial decrease in data search and retrieve time after introducing B-tree based indexes.

Table 5. Database performance in terms of execution time.

Entity (Size)	Total Records	Fetched Records	Trials Count	Exec Time (ms)	Fetch Criteria (Event, Actor, Source)
amc_data (28 GB)	$164x10^6$	2,581	1	202	T-pose, bd, HDM05
		237,771	2	8,134	Throwing, mm, HDM05
		751,042	20	24,576	Walking, bd, HDM05
		1,505,390	13	51,182	Dancing, All Actors, HDM05
		126,701	12	4,293	Walk, 07, CMU
		522,058	19	17,656	Modern Dance, 05, CMU
		744,662	7	26,081	Swimming, 125, CMU
c3d_data (32 GB)	$230x10^6$	360,756	2	15,978	Throwing, mm, HDM05
		1,139,512	20	48,121	Walking, bd, HDM05
		2,196,786	13	98,768	Dancing, All actors, HDM05
		179,129	12	8,395	Walk, 07, CMU
		738,082	19	34,928	Modern Dance, 05, CMU
		1,052,798	7	47,209	Swimming, 125 , CMU

The performance of a database can be analyzed based on how much time it takes to search and retrieve records against simple and complex queries. As said earlier, we have a particularly large database with a disk size of approximately 61 GB. The two

largest entities in our database are *'amc_data'* and *'c3d_data'* having a disk size of *28 GB* and *32 GB* respectively. In section 5.1, we have outlined our strategy to optimize performance of the two entities by means of indexing. To test the performance of the database, we have executed several queries on these two large entities. In general, we have found minimum search and retrieval time when the retrieved collections are small in count and maximum search and retrieval time when the retrieved collections are large in count.

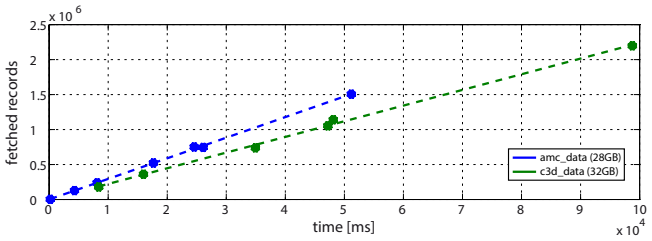

Fig. 2. Scatter plot of the timings, when querying the AMC and C3D datasets. We observed a linear relation between the number of retrieved records and the execution time.

In order to test the performance of the database, several fetch criteria are used to fetch data and the results are presented in Table 5. All tests are performed locally on the database server machine. The database took only *202* ms to fetch *2581* records of *'T-pose'* data of the actor *'bd'* for *HDM05*. The database took *4293* ms to fetch *126701* records of the *'Walk'* event of the actor *'07'* (12 trials in total) from *CMU*. One the other hand, it took *51182* ms to fetch *1505390* records of the *'dancing'* event for all actors (13 trials in total) from *HDM05*. During experimentation, we have observed that the execution time increases as the size of the retrieved records increases and a linear tendency is seen as shown in Figure 2. From this, we conclude that the performance of the database is optimal for small record sets. Most applications work in cycles of retrieving small chunks of data from the database and processing these records instead of retrieving the whole data at once. With small execution time (such as *202 ms*), it is possible to achieve interactive processing by fetching and processing data frame by frame.

Complexity Analysis of SQL Queries The complexity of an SQL statement depends upon a number of factors such as number of tables involved, number of joins, unions, intersections, where/having clauses, sub-queries and so on. The complexity of an SQL statement directly effects its execution cost. The execution plan of any SQL statement can be analyzed in PostgreSQL using the '*explain*' command. "The execution plan shows how the table(s) referenced by the statement will be scanned by plain sequential scan, index scan, etc. and if multiple tables are referenced, what join algorithms will be used to bring together the required rows from each input table" [27].

Retrieving Motion Information of an Event: The SQL query of *retrieving motion information of an event* is given in number 4 of the section *Retrieving Collections* 4.2.

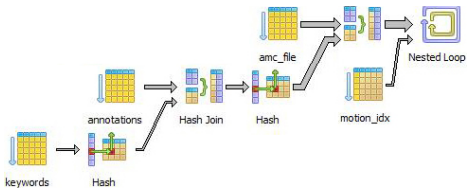

Fig. 3. Complexity analysis of the query 'retrieving motion information of an event' analyzed by PgAdmin Query Tool. This query involves inner-joins between four entities: 'keywords', 'annotations', 'amc_file', and 'motion'. It took 12 ms to fetch 13 records.

A visual complexity analysis of this query is presented in Figure 3. This query retrieves motion IDs of all motion records for the '*dancing*' event. It consists of four inner-joins between entities: '*motion*', '*annotations*', '*amc_file*', and '*keywords*'. In order to relate entities, PostgreSQL uses indexes for those entities which are indexed and hash joins are used for non-indexed entities. In this example only '*motion*' entity is indexed so its index (*motion_idx*) is used to retrieve records. This query took *12 ms* to fetch *13* records.

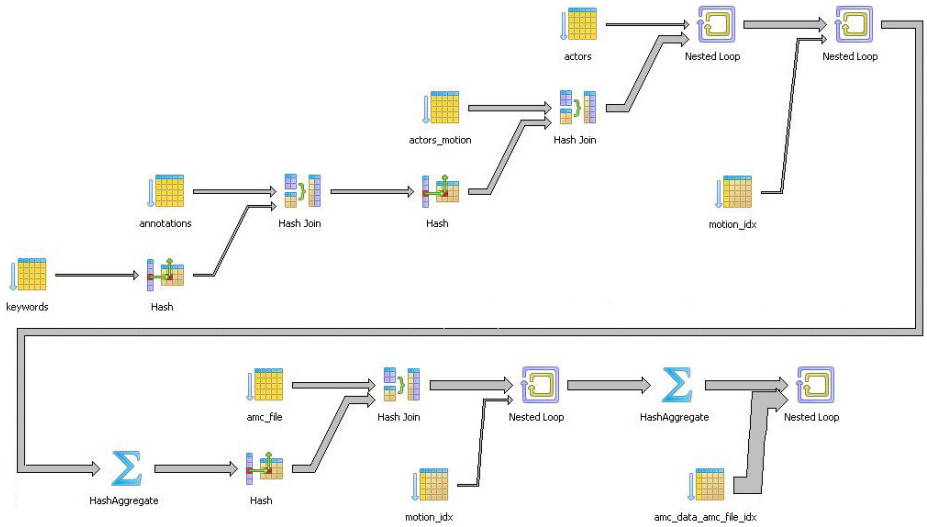

Fig. 4. Complexity analysis of the query 'retrieving AMC data' analyzed by PgAdmin Query Tool. This query consists of two sub-queries and seven inner-joins between entities: 'keywords', 'annotations', 'actors_motion', 'actors', 'motion', and 'amc_file'. It took 8,066 ms to fetch 237,771 records.

Retrieving AMC Data: The SQL query of *retrieving AMC data* is outlined in number 6 of the section *Retrieving Collections* 4.2. This is one of the most complex queries in our

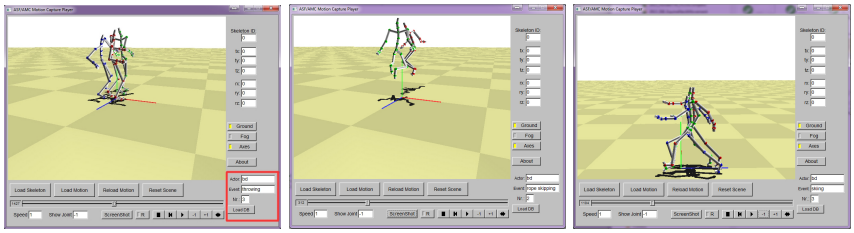

Fig. 5. Extended version of ASF/AMC motion capture player originally developed by [28]. This extended version can be used to fetch data from the database and play it. In the left side figure, the rectangle (right side bottom) highlights the extended part. The user provides as input an 'actor name', an 'event', and an optional 'motion number'. The example shows three 'throwing' actions (left), two 'rope skipping' actions (center), and three 'skiing' actions (right). All motions are performed by the actor 'bd'.

schema. A visual complexity analysis of this query is presented in Figure 4. This query retrieves AMC data records of all '*throwing*' events performed by the actor '*mm*'. This query consists of two sub-queries and seven inner-joins between entities: '*keywords*', '*annotations*', '*actors_motion*', '*actors*', '*motion*', and '*amc_file*'. Entities '*motion*' and '*amc_data*' are indexed and their indexes (motion_idx, *motion_data_amc_file_idx*) are used to retrieve records. The query took *8,066 ms* to fetch *237,771* records. This execution time is fairly acceptable considering the size of the entity '*amc_data*' (approx. *90 million* records) and the complexity of this query which involves two sub-queries and multiple inner-joins.

6 Applications

6.1 Extended Motion Capture Player

An ASF/AMC motion capture player is one of many ways to visualize motion frames as animation sequence. For this purpose skeleton and motion data for each frame are required. A basic ASF/AMC motion capture player is available with the CMU motion capture dataset [28], which reads skeleton and motion data from flat files and plays them as an animation. We present an extended version of this motion capture player which is capable of motion import from our database. A new GUI element was added, where the variable parts of the *'retrieve AMC data'* SQL query (see Sec. 4.2) can be filled. As input an *'actor'*, an *'event'*, and an optional *'motion number'* can be given. A database search is carried out for the specified input parameters and if the data is found, it is loaded into the player. A user can then use various control buttons provided in the player to play the animation. Multiple motions can be loaded and played at the same time. With this type of interface it is simple to search for individual motion sequences without having knowledge, or even touching the actual motion capture files. Figure 5 shows three different types of motions loaded in the extended motion capture player. In the left side figure, the rectangle (right side bottom) highlights the extended part. The example shows three *'throwing'* actions (left side figure), two *'rope skipping'* actions (center figure), and three *'skiing'* actions (right side figure). All motions are performed by the actor *'bd'* in these examples. This simple example already shows how

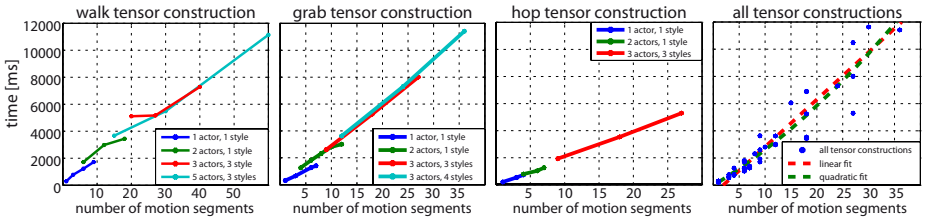

Fig. 6. Scatter plot of the timings, when querying the amc dataset to construct motion tensors. We observed a linear relation between the number of fetched motion segments, but no dependency in the number of actors, motion classes or styles.

simple selections can be made on the basis of SQL queries. More sophisticated data visualization techniques [22, 29] could use such selections to allow rapid drill down to important parts of motion data sets for further exploration and analysis.

6.2 Automatic Generation of Statistical Models

Another set of techniques that can benefit from the connection to a relational mocap database is the automatic construction of statistical models. Such models are used for analysis tasks such as motion classification and segmentation or motion synthesis tasks where motion sequences should be developed that fulfill certain user specified constraints. To show the effectiveness of our approach, we show the automatic construction of motion tensors, that have been shown to be useful for motion synthesis [7, 8]. To this end, we fetched data from the database for various actors in the same number of repetitions for multiple motion classes that belong to various styles of a motion. Krüger et al. [7] introduced so called *natural modes*, that belong to different actors, repetitions or motion styles. In the original works these example motions were selected by hand carefully. By using a data retrieval function, which is written in procedural language for the PostgreSQL (PL/pgSQL), we fetch the individual motions for construction of the multi-modal model. The function takes as input *actor name*, *motion class*, and *number of repetitions* and retrieves related data from the database. Using this approach the construction of each tensor model, as described by Krüger et al. [7], needed less than ten seconds. Larger sets of motions, including up to 5 actors and 4 motion classes could be retrieved in about 12 seconds. The actual motions for tensor construction where taken from the HDM05 motion capture database. For the *walk*-tensor examples motions from the motion classes: *walkRightCircle4StepsRstart*, *walk4StepsRstart*, *walkLeftCircle4StepsRstart* were used. For the *grab*-tensor the classes *grabHighR*, *grabMiddleR*, *grabLowR*, *grabFloorR* were retrieved. And for the *hop*-tensor the classes *hopRLeg2hops*, *hopBothLegs2hops*, *hopLLeg2hops* where used. The annotations from the classes where taken from the so called *cut-files* subset which is described in the documentation [2] of the HDM05 data set. Overall we observed that retrieval times depend linear on the number of fetched motion segments instead of the number of actors or motion classes (See Fig. 6). Thus, large data sets can be the basis for an efficient construction of statistical models and therefore for a bunch of new applications in motion analysis and synthesis.

7 Conclusion and Future Work

In this paper, we presented a relational database scheme for mocap data sets. According to this scheme, a database has been created using the open source PostgreSQL RDBMS, and motion capture data from HDM05 and CMU datasets. The functionality has been shown in two applications: A simple player where specific motions can be loaded without touching the actual files and the construction of motion tensors as example for statistical models, which can be used for further motion analysis and synthesis steps.

The proposed database can easily be extended to derived entities of the motion capture data sets. Thus, more complex annotations, physics based features or new sensor modalities (Videos, Accelerometers, Gyroscopes, etc.) are easy to incorporate. To combine further, more complex data-driven methods with the presented database setup is planned for future research.

References

1. CMU: CMU Motion Capture Database. http://mocap.cs.cmu.edu/ (2003)
2. Müller, M., Röder, T., Clausen, M., Eberhardt, B., Krüger, B., Weber, A.: Documentation Mocap Database HDM05. Technical Report CG-2007-2, Universität Bonn (June 2007)
3. Sigal, L., Balan, A., Black, M.: Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. International Journal of Computer Vision **87**(1-2) (2010) 4–27
4. Guerra-Filho, G., Biswas, A.: The human motion database: A cognitive and parametric sampling of human motion. In: Automatic Face Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on. (March 2011) 103–110
5. Zhou, F., la Torre, F.D., Hodgins, J.K.: Hierarchical aligned cluster analysis for temporal clustering of human motion. IEEE Trans. on Pattern Analysis and Machine Intelligence **35**(3) (2013) 582–596
6. Vögele, A., Krüger, B., Klein, R.: Efficient unsupervised temporal segmentation of human motion. In: 2014 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. (July 2014)
7. Krüger, B., Tautges, J., Müller, M., Weber, A.: Multi-mode tensor representation of motion data. Journal of Virtual Reality and Broadcasting **5**(5) (July 2008)
8. Min, J., Liu, H., Chai, J.: Synthesis and editing of personalized stylistic human motion. In: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. I3D '10, New York, NY, USA, ACM (2010) 39–46
9. Min, J., Chai, J.: Motion graphs++: A compact generative model for semantic motion analysis and synthesis. ACM Trans. Graph. **31**(6) (November 2012) 153:1–153:12
10. Baumann, J., Wessel, R., Krüger, B., Weber, A.: Action graph: A versatile data structure for action recognition. In: GRAPP 2014 - International Conference on Computer Graphics Theory and Applications, SCITEPRESS (January 2014)
11. Kovar, L., Gleicher, M.: Automated extraction and parameterization of motions in large data sets. ACM Transactions on Graphics **23**(3) (2004) 559–568 SIGGRAPH 2004.

12. Lin, Y.: Efficient human motion retrieval in large databases. In: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia, ACM (2006) 31–37
13. Basu, S., Shanbhag, S., Chandran, S.: Search and transitioning for motion captured sequences. In: Proceedings of the ACM symposium on Virtual reality software and technology, ACM (2005) 220–223
14. Liu, G., Zhang, J., Wang, W., McMillan, L.: A system for analyzing and indexing human-motion databases. In: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, ACM (2005) 924–926
15. Müller, M., Röder, T., Clausen, M.: Efficient content-based retrieval of motion capture data. In: ACM Transactions on Graphics (TOG). Volume 24., ACM (2005) 677–685
16. Forbes, K., Fiume, E.: An efficient search algorithm for motion data using weighted pca. In: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, ACM (2005) 67–76
17. Arikan, O., Forsyth, D.A., O'Brien, J.F.: Motion synthesis from annotations. ACM Transactions on Graphics (TOG) **22**(3) (2003) 402–408
18. Krüger, B., Tautges, J., Weber, A., Zinke, A.: Fast local and global similarity searches in large motion capture databases. In: 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. SCA '10, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association (July 2010) 1–10
19. Wang, P., Lau, R.W., Zhang, M., Wang, J., Song, H., Pan, Z.: A real-time database architecture for motion capture data. In: Proceedings of the 19th ACM international conference on Multimedia, ACM (2011) 1337–1340
20. Awad, C., Courty, N., Gibet, S.: A database architecture for real-time motion retrieval. In: Content-Based Multimedia Indexing, 2009. CBMI'09. Seventh International Workshop on, IEEE (2009) 225–230
21. Ramanan, D., Forsyth, D.A.: Automatic Annotation of Everyday Movements. In: Neural Information Processing Systems. (2003)
22. Bernard, J., Wilhelm, N., Krüger, B., May, T., Schreck, T., Kohlhammer, J.: Motionexplorer: Exploratory search in human motion capture data based on hierarchical aggregation. IEEE Trans. on Visualization and Computer Graphics (Proc. VAST) (2013)
23. Lin, E.H.: A research on 3d motion database management and query system based on kinect. In Park, J.J.J.H., Pan, Y., Kim, C., Yang, Y., eds.: Future Information Technology - II. Volume 329 of Lecture Notes in Electrical Engineering. Springer Netherlands (2015) 29–35
24. Xsens: Products - Xsens. http://www.xsens.com/en/general/mti (July 2013)
25. Robbins, K.L., Wu, Q.: Development of a computer tool for anthropometric analyses. In: Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences (METMBS03), pages 347353, Las Vegas, CSREA Press (2003)
26. PostgreSQL: Documentation: 9.0: Index Types. http://www.postgresql.org/docs/9.0/static/indexes-types.html (July 2013)
27. PostgreSQL: Documentation: 9.0: EXPLAIN. http://www.postgresql.org/docs/9.0/static/sql-explain.html (July 2013)
28. Barbič, J., Zhao, Y.: ASF/AMC Motion Capture Player. http://mocap.cs.cmu.edu/tools.php (July 2013)
29. Wilhelm, N., Vögele, A., Zsoldos, R., Licka, T., Krüger, B., Bernard, J.: Furyexplorer: Visual-interactive exploration of horse motion capture data. In: Visualization and Data Analysis (VDA 2015). (February 2015)