Connectionist Databases

Before describing the unique features of Connectionist Databases, I have summarised the main features of the main other database systems for comparison purposes.

Relational Databases

For a long time, the most widely used type of database has been the Relational Database Management System (RDBMS). The concept was proposed by an IBM employee, E.F. (Ted) Codd in 1970. Oracle, the first commercially available RDBMS package, was released in 1979. IBM developed their own RDBMS which was eventually called DB2. They also developed an RDBMS related query language which they called SQL (Structured Query Language).

Relational databases have serious problems with scalability and performance, particularly in the 21st century when data volumes have expanded at an extraordinary rate. RDBMSs tend to slow down quite considerably as file sizes grow. Response times depend too much on the way that the databases are designed. Certainly, they cannot cope with the extraordinary transaction rates and enormous data volumes required for many modern applications.

One of the biggest problems of creating any application with relational database management systems is designing the database. The process of 'normalising' data for RDBMS is intended to create a database design that provides best performance with least redundant data. This takes specialist skills and experience and the result is always a compromise, at best valid only at design time. Response times tend to increase unacceptably as file sizes grow. Different database experts will produce a different design.

NoSQL Databases

Some non-relational databases are known today as NoSQL databases although the acronym didn't become popular until the early 21st century. Many non-relational databases existed before then, so it is not possible to determine which was the first. Despite the acronym, SQL can be used to query some NoSQL databases.
,
Whilst there is a standard definition of RDBMS, this does not exist for NoSQL databases and there is no industry wide list of mandatory features.

Modern NoSQL databases are designed to handle the problems of RDBMS, but they have their own disadvantages, in particular a possible lack of consistency and difficulties with transaction processing.

Hierarchical Databases

In a hierarchical database, data are considered as a tree-like structure with each parent having a unidirectional relationship with one or more children. But, in a family tree, if Fred has a child Joe, then, rather obviously, Joe has a parent Fred. Thus the relationship between parent terms and child terms is really bidirectional rather than unidirectional but the structure of a standard hierarchical database is not able to reflect this. This means that to find the parent of any child term, the whole database has to be read, with serious performance repercussions.

As is explained below, relationships are always bidirectional in a Connectionist Database, which handles hierarchies without problem and with immediate responses, without the need to read the whole database. The ability of the ERROS Connectionist Database to store effectively unlimited levels of hierarchy is described later.

Network Databases

While the hierarchical database model structures data as a tree of records, with each record having just one parent record and many children, network databases allow any record to have both multiple parents and multiple children. However, the relationships defined are still unidirectional so that users cannot navigate from any record to any other connected record. The bidirectional relationships of the Connectionist Database allow users to navigate all connections in either direction, limited only by their security. In the ERROS Connectionist Database, all metadata and user data is stored in nodes in a single network database as described later. All connections between the nodes are always bidirectional.

Connectionist Database

A fifth type of database is the Connectionist Database. This is quite different from all other databases, such as Relational and NoSQL databases. Only one implementation is known – the ERROS Connectionist Database – so that becomes the de facto standard and its features are described below.

The ERROS Connectionist Database was developed in 1981/82 with two main goals -

1. To meet a need for a scalable database that could handle many terabytes of data, although a few gigabytes was thought to be a lot of data at that time.

   The ERROS Connectionist Database has very high scalability, with almost zero second response times that do not noticeably deteriorate as data volumes grow. To a considerable extent, this has been achieved by abolishing the need for physical database design, thus eliminating human involvement and judgement, and so ensuring total control over performance, with consistent, predictable results.

2. To provide a way of developing complex computer applications incrementally so that changes to the database structure of an application could be made without the need to shutdown and reprogram the application as was necessary with iterative development.

   The emphasis was on Application Development rather than Software Development. It seemed obvious that, when human beings learn something new, their brains are not shut down, redesigned, reprogrammed, tested and restarted – they learn on the fly.

   This simple insight suggested that, if the database structure and much of the program logic were defined in the database, rather than in program code, then updating the database would allow applications to be changed on the fly and enable genuinely incremental development without the need to shut down the application.

The name is based on the ability of the ERROS Connectionist Database to associate or relate any data through myriad connections, in which capability it apparently has some similarity to the brain. These connections can be temporal – date and time dependent.

An important design decision was to store all data in a small, fixed number of identical files. These contain interchangeable, relatively short, fixed length records, which are all updated and accessed by a single database handler. No new files are created when new entity types are defined. Having very few files ensures optimum performance and makes implementation of automatic concurrency control straightforward.

Whereas Relational Databases store data in rows within tables, with a record for each row, the ERROS Connectionist Database uses a separate record to store each attribute or cell (i.e. row/column). Where the data is longer than that record, multiple records can be used. The absence of any records in a cell means that there is no data in the attribute, so that totally variable length records are possible, without the need for null values. With relational

databases, fixed length records with null values in empty fields waste a lot of space and can badly affect performance. The ERROS design also allows multiple entries in the same cell so that there can be repeating attributes, or repeating fields where there is only one field in the attribute. Storing each attribute iteration in separate records minimises contention problems.

The database structure described above is quite different from other databases, yet the most important distinction is that the ERROS Connectionist Database also stores data and application definitions in the same database as the user data that these define, rather than in their traditional location in computer programs, so allowing totally incremental application development simply by updating the database, without program changes most of the time. Such changes can be made whilst an application is live, without affecting the response times of any operators using the application.

Bidirectional Relationships

Equally importantly, and also apparently uniquely, the ERROS Connectionist Database can store bidirectional relationships that can be navigated in either direction with the same instant response times, limited only by security. No query language is required and there are no joins.

A vital feature of the ERROS bidirectional relationships is that the term defining the relationship can be different in each direction. Taking again the example of the family tree, if Fred has a child Joe, then Joe has a parent Fred. The term "child" only defines the relationship from the viewpoint of Fred. The term "parent" only defines the relationship from the viewpoint of Joe. In this case, "child" is the inverse of "parent". Of course, Fred will have had parents and Joe may have his own children. There will be occasions when the term defining the relationship will be the same in both directions – for instance if Fred loves Joe and Joe loves Fred. The Connectionist database allows unlimited separately defined relationships between the same two entities. These are defined at the entity type level and the definitions are inherited by the individual entities. A visual representation of the structure of ERROS bidirectional relationships can be found under the heading "ERROS Semantic Network Diagrams".

The majority of the data and application definitions and much of the user data are established as bidirectional relationships or connections with other definitions or data. These connections are stored in the database as are the definitions of the connections that are valid.

Applications, defined in the database, act as filters and provide access paths to the data. ERROS applications are not compiled and the concept of the compiled, rigid, inflexible applications created by software development has gone. The database handler interprets the stored data and application definitions at very high speed. Application development is achieved without any new files being created. Creating new data definitions or applications does not generally require new programs or program changes, but any such changes will be in 'exit' programs and will not affect the main ERROS programs. ERROS has its own trigger mechanism so no special trigger programs are required.

All aspects of a user's data are stored in a central business model, defined in terms of entity types, their attributes, and the connections between them. The user's natural language (English, French, German etc.) is used for these definitions rather than a programming language that the user may not understand. Entity types, entities, attributes and applications are automatically given numbers and these form part of the multi-element key field of the files, as does the cell data record identifier – e.g. name or number or by date and time. These identifiers are compressed using two stage ERROS algorithms that have been patented. Developers and users can retrieve any data by full or generic name as well as by number.

The unique, advanced 17 part binary key field allows direct access to the data in any cell. Records in cells can be dual indexed – e.g by name and also by number, - so that records in the attribute employee can be retrieved by employee name or employee number and can be displayed or printed in both name sequence – stored as last name, first name, although they

can be displayed as first name, last name - and also in employee number sequence. Sales or purchase order lines in an order can be displayed in order of product name and also product number. Sales orders for each product can be displayed in date sequence or customer name sequence. No program coding or sorting is required. Records can be retrieved instantly by name or number without searching, even in orders with a very large number of lines.

There is little distinction between metadata and user data and all records are accessed and updated by the same database handler. When you are using the ERROS Application Creator and you define the valid entity types and their attributes, create menus, etc. for use in another application, the database updates that you make are your user data in the Application Creator. But, when you operate the application that you have created or changed, those same records are now your metadata, controlling what you can do in that application. The ERROS Connectionist Database is defined by itself in itself, using the same structure.

Rather than being accessed by application programs, or by an API, the ERROS Connectionist Database is part of a complete development and operational environment, ERROS, which provides a graphical interface that generates HTML and JavaScript on the fly and can be operated using a browser over the internet. ERROS also generates PCL 5 for printing. ERROS has its own Output Description Language, allowing the creation of complex typeset documents that can combine both text and user data stored in the database. ERROS can generate XML.

ERROS Menus

ERROS application menu records, stored in the database, determine which entity types are accessible in each application, which attributes are available for each entity type in that application and the action(s) that can be undertaken for each attribute. Nested menus allow complex procedures to be defined in the database. Menus also define boundaries for transaction processing, and enable constraint processing and ensure referential integrity.

Menus define the views of the data, including the contents of rows. The user interface allows multiple attributes from multiple entity types, including images, to be displayed together in the same row. There can be any number of separately defined views of the data for display and also for printing.

Access to the data in any attribute or cell is always through a menu record. This means that, for every attribute or cell for an entity type in every application, there is a menu record that controls the way it is processed and presented. The menu records control the action to be taken when there is only one attribute iteration or when one has been selected, e.g. update, or navigate to the next link in the network. This provides extraordinary flexibility and different menus can be used for the same attribute in different parts of the same application or in other applications. Default values are normally used in menu records so that creating them is very simple most of the time.

There can also be functional and environment menus. These optional menus allow special menus to be selected depending on the action taken on the previous data record, such as added, changed, marked for deletion (i.e. hidden) or removed. They are mostly used for transaction processing or different views of data for printing.

There can be both public and private user menus within an application. ERROS applications can be set up to look for a user version of a menu, but, if not found, to access the public version. This facility allows complex collaborative applications, such as ERP, to be created, and, using user menus, modifications can be made for a particular ERROS user company without the need to change the original package. Menus can be nested, with an unlimited number of levels.

Physical Record Layout

Each record in a cell (i.e. each attribute iteration) has a key field, a control field, and a unique data field that contains metadata or user data, stored in a Universal Data Type. The control field includes a transaction number, related to a time stamp, and also defines the layout of the meta or user data. Records can contain metadata or user data or text, or URLs or other identifiers of videos, still images, audio and other files. This allows ERROS users to index web pages with words, names or phrases that are not actually contained in the pages. They can also use ERROS to index videos, images, PDFs, etc. Those cells that contain a connection also have a link field, invisible to developers or users. This is used for direct navigation to the connected record. The multipart key field allows direct access to any attribute iteration in any cell, without a query language. Users and developers are unaware of the key field, control field and link field.

The database structure allows multiple record layouts to be stored in the same table. This means that the ERROS Connectionist Database can also be described as a wide column store – see later. It is even possible to have multiple record layouts in the same cell.

The database can combine very complex multidimensional, relational, hierarchical and network data structures. All ERROS applications are automatically integrated, sharing all data without redundancy.

Audit Trail

A complete audit trail records who changed any metadata or user data, the application used and the date and time, together with the transaction number of the last transaction that modified the record. Every change to both metadata and user data is logged, with before and after images being recorded, and this cannot be bypassed. There will not be any gaps in the transaction numbers, even if a transaction does not complete, perhaps having been rolled back after failing a constraint test, or due to a power or hardware or software failure.

ERROS application development is largely self-documenting. All changes to all metadata and all user data can be rolled back.

The ERROS Connectionist Database has outstanding security which can be set at the level of individual attributes or cells or even on individual instances in a cell – see http://www.erros.co.uk/ERROS_security.htm. The complex structure of the binary key field means that the database cannot be accessed or updated by any utilities or query languages such as SQL. Only the ERROS database handler can access or change the database.

The ERROS Connectionist Database was designed, from the very beginning, to ensure that data, once entered, can never be lost without trace and that all data can be totally relied upon to be there - in perpetuity. - see http://www.erros.co.uk/ERROS_Connectionist_Database_Perpetual_Data.pdf

An ERROS high availability option allows multiple local or remote duplicate servers to maintain up-to-date copies of all metadata and user data. Providing there are no communication problems, these should be no more than a fraction of a second behind the prime system, and one of them could be switched to become the prime system if that fails. Users who simply require read only access could use one of the duplicate servers, so reducing the load on the main system.

Most attribute values are stored as bidirectional connections. For instance, an entity type 'Telephone Number' can store records for each telephone number known to a user company, If the contacts, both personal and business, are related to their telephone numbers, then, as ERROS can store many-to-many relationships and because all such connections are automatically bidirectional, accessing a telephone number in the entity type Telephone Number will cause ERROS to display the name(s) of the related contact(s). No query language is required and the result will be instant. Information about each contact can be displayed

alongside the names, with the definition of the view required stored in the database.

Unlimited, nested, lower level connections can be made. ERROS also allows unlimited nested hierarchies. “Children” can have multiple “parents” and users can travel up or down hierarchies and enter and exit at any level, not simply at the top. Data can be stored about any connection or hierarchy in a surrogate record and accessed automatically from either side of a connection without needing to navigate to it. The database can handle groups or sets and is ideal for ontology and taxonomy.

The automatic indexing of all records and attribute iterations, together with the link field to connected records, means that the raw data is turned into a fully navigable semantic network containing all of a user's data, with security determining how much of this users can see in each application.

ERROS was designed for collaborative computing and the attributes in any logical record may contain both 'public' and 'private' (i.e. user) data. Users retain secure control of their data.

Any entity can belong to multiple entity types and inherit the attributes of the entity type under which it is being considered. Data in those attributes can be stored in either the main or in a secondary entity type. If the XYZ Company is entered initially as a contact name, with address, telephone number, etc. that data would be stored in the main contact entity type. If the company then became a customer, and was being considered as such, the menu for customer would also contain address and telephone number, as well as orders received, etc. But, if the customer has only one business address, then that would not need to be entered again as, when displaying the customer details, the ERROS application could be instructed, using a simple procedure stored in the database, to look first for the address in the customer entity type, but, if there is none, to retrieve it from the main entity type.

Although human beings have no idea of the structure of the data in their brains and certainly do not have a picture of any of the data, they cope perfectly well without this. Developers do like to see images explaining the structure of the data in their applications. Data are multidimensional whereas images of a data structure are two dimensional and, in any but the simplest application, the data structure is very complex, and impossible to illustrate or even understand as a whole.

ERROS can display or import data from other sources or systems (programming in an 'exit' program, that does not affect the ERROS main programs, may be required). ERROS can also export XML files.

Grow-as-you-go

The ERROS Connectionist Database can be changed at any time by users with appropriate security, and storing definitions in the database rather than in program code allows fully incremental development, without detailed upfront user and system specifications and without physical file design. Adding a new attribute to an existing entity type whilst that is in use in an application has no impact on the performance of any users on line at the time, and the application will not be affected unless it is changed to include the new attribute. This can also be done whilst the application is live. Changing database records is dramatically faster and very much more productive than even the most simple changes to program code.

ERROS Semantic Network Diagrams

In ERROS, developers can create the most complex data structure without needing any picture of the data. However, the structure of the data linking any two nodes in the semantic network that stores the data can be represented in an ERROS Semantic Network Diagram. These can be used to illustrate any part of the structure of data or applications and two examples are shown below. In any connection, an entity type (Entity type A) has an attribute (attribute of entity type A) and this is a relationship with another entity type (Entity Type B). Both entity

types are identified by name and/or number. Entity type B has an attribute (attribute of entity type B) which is a relationship with Entity type A. The two attributes are the inverse of each other. That does not necessarily mean opposite. For instance, a boy may love a girl, but the girl may not be aware of his existence, or may be indifferent to him or may dislike him or may also love him. Data can be stored about the relationship or connection in a surrogate record.

In figure 1, an entity type named 'Entity Type' has an attribute 'attributes include'. This is defined in the Connectionist Database as being a relationship with an entity type named 'Attribute'. That has an attribute 'attribute of' which is defined as a relationship with the entity type 'Entity Type'. 'attribute of' is the inverse of 'attributes include'. This diagram shows part of the structure of the Connectionist Database itself – showing how it is defined by itself in itself. Information about the characteristics of each attribute for each entity type are put in a surrogate record.

ERROS Semantic Network Diagrams

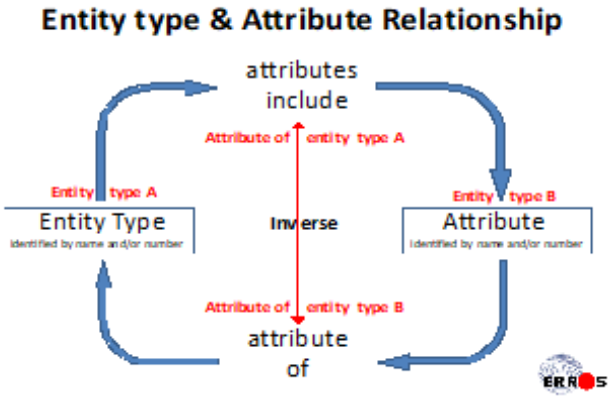

Fig. 1

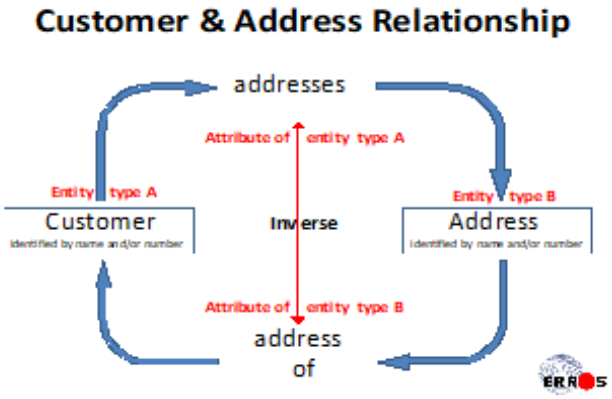

Fig. 2

Figure 2 shows how customers are connected to addresses. Each address known to the ERROS user company is stored as a separate record in the entity type 'Address'. Customers are connected to addresses. Many-to-many connections are possible with ERROS so more than one company can be at an address and any customer can have multiple addresses. The address records are dual indexed – by zip or postcode and by state/county, town, street, etc. In addition, the application could be set up to access streets directly by name with a list, in number sequence, of all recorded addresses in the street.
#
In the ERROS Business Modeller application, you can see, for each entity type, the attributes that have been defined. Some attributes, such as address, are appropriate for many entity types, such as Customer, Employee, Supplier, etc. The characteristics of address only need to be defined once and, because ERROS is object oriented, the entity types will inherit those when they are connected to the attribute although any characteristic can be overridden. To find out which entity types have the attribute Address, access the term in the entity type Attribute and you can view a list of the entity types. Double click on any of those to navigate the connection and you can see all the attributes of the entity type that you have selected.

Entity type and attribute names are in the natural language of the user and spaces are allowed between words in the names – there is no need for 'customername' or 'customer_name' or, even worse, 'CUSNAM'.

ERROS based applications have much lower Total Cost of Ownership for development, maintenance and operations. The speed and ease with which the ERROS Connectionist Database allows application development and maintenance means that users can get what they want when they want it.

STIPPLE

The ERROS Connectionist Database was created in order to create STIPPLE, a very complex collaborative application for recording fine and applied art and other history. Nothing like this had been attempted before and it was obvious that the traditional iterative application development method of detailed user and system specifications, detailed file designs and then program coding, would not work. It would only be possible to create STIPPLE if this could be done incrementally. There was no application development method at that time that allowed incremental development. As a result, the decision was made that the first step in creating STIPPLE had to be the creation of a new application development tool that allowed incremental development. If this was not possible, then STIPPLE would not be created. But ERROS and the ERROS Connectionist Database were successfully created and went live in 1982, with STIPPLE going live in February 1983. STIPPLE has continually evolved over the years. For information about STIPPLE, go to http://stipple.co.uk/STIPPLEWEBA/Stipple_Home.htm.

Wikipedia https://en.wikipedia.org/wiki/Incremental_build_model refers to an incremental build model but this is about repeated iterations of the waterfall model that are applied incrementally. This cannot be described as genuine incremental development. Every iteration will require at least some program compilation. If, during testing, a field is found to be missing from, say, the customer file, then, any applications using that file will need need to be shut down so that the file can be changed. Some or all programs that access that file may need to be changed and all will need recompiling.

In ERROS, a new attribute can be added to a file, and to any browser screen or printed report at any time, without shutting down the system and these changes will immediately be internet enabled.

ERROS allows continuous application development, whilst an application is live and without program compilation. Today the only truly incremental application development system is ERROS. It is a total paradigm shift in both database structures and modern application development methods so cannot easily be compared with traditional development methods nor can it be described in a few sentences.

ERROS has many concepts that may be unfamiliar. It is not like any other product. Yet it is very simple. The ERROS main programs, including the database handler, GUI, typesetting and other features are provided by just four main programs totalling about 17mb of code.

ERROS runs under the IBM i operating system on the IBM Power Systems platform. To learn how ERROS works, visit http://www.erros.co.uk/ERROS_Description_2014-08-16.pdf

Memex

One of the most forward looking and influential scientific thinkers of the 20th century was Dr. Vannevar Bush, an American engineer, inventor and science administrator who, during World War II, was in charge of almost all US military research. He was responsible for the initiation and early administration of the Manhattan Project.

Bush conceived the Memex, a hypothetical hypertext system which he described in his 1945 *The Atlantic Monthly* article "As We May Think". Bush envisioned the Memex as a device in which individuals would compress and store all of their books, records, and communications, "mechanized so that it may be consulted with exceeding speed and flexibility". The Memex would provide an "enlarged intimate supplement to one's memory". The technology required to create such a system did not exist at that time but Bush's concept of the Memex influenced the development of early hypertext systems (eventually leading to the creation of the World Wide Web).

Bush envisaged Memex mimicking the way the human brain links data by association, travelling the connections, and also allowing the storage of personal annotations. He did not consider the use of metadata to provide structure for the data.

Bush envisaged Memex as being collaborative, with the ability to connect, annotate and share data.

Although created without prior knowledge of Memex, the ERROS Connectionist Database might be considered to be an implementation of Memex, with the addition of metadata and advanced indexing. ERROS has the ability to create collaborative applications with the capabilities that Bush described, with unlimited associations or connections, and the unique power that allows these to be navigated in either direction at the same very high speed. One such collaborative application, created using ERROS, is STIPPLE, described above.

Wide Column Store

For wide column store, see https://en.wikipedia.org/wiki/Wide_column_store. This says "A **wide column store** is a type of NoSQL database. It uses tables, rows, and columns, but, unlike a relational database, the names and format of the columns can vary from row to row in the same table. A wide column store can be interpreted as a two-dimensional key-value store. Data is stored in a row-by-row fashion with the columns for a given row stored together."

Only a few wide column stores are listed. One of these is Google's Bigtable database.

Google had special problems with obtaining acceptable performance from vast volumes of data and, in 2004, they commenced development of their Bigtable database - see https://en.wikipedia.org/wiki/Bigtable. Google's reasons for developing its own database included scalability and better control of performance characteristics.

The ERROS Connectionist Database, which is also a NoSQL database, equally allows different record layouts in the same table and therefore may also be described as a wide column store, although it existed over twenty years before Bigtable or the name Wide Column Store. However, in ERROS, each attribute instance in each cell is stored in a separate record, with a time stamp, rather than on a row-by-row basis. Both ERROS and Bigtable have row keys, but ERROS also has an advanced multipart key field for each cell data instance.

Whereas Google's aim was, according to Wikipedia, for better control of performance, the aim of the ERROS design was for **total control** of performance and this has been achieved, with consistent, almost zero second, response times that do not noticeably change as data volumes expand. Wikipedia describes Bigtable as a "one of the prototypical examples of a wide column store", although ERROS had existed for twenty two years before Google started development of Bigtable.

Google's Bigtable can handle petabytes of data spread over many machines. ERROS can handle many terabytes of data, in addition to images, documents etc., which will be more than adequate for most very large companies. ERROS also provides a totally new type of database that allows truly incremental application development with outstanding performance, resulting in much lower total cost of ownership.

ERROS was conceived and created by Rob Dixon who patented the concepts of ERROS. He used ERROS to create STIPPLE.

Rob Dixon – ERROS, 2nd January 2021
rob.dixon@erros.co.uk