Relational Databases

An Overview of RDBMS, Client/Server, SQL and all that

Presented by Courts Carter

To adequately cover a topic such as "Relational Databases" requires many lengthy, but interesting digressions. This is because Relational Database Systems are but one component in the larger discipline of systems design. To achieve success with the database portion one must judiciously apply a working knowledge of these:

- Information Engineering
- Client /Server system design
- Relational Database Management System
- Structured Query Language

You just won't hear someone mention one of the above without an implied reference to the other three, partly because the borders between these are quite fuzzy. The fact is, they all had to mature to a critical point before their collective power could be realized and dubbed a success; a synergy which did not happen until the tail-end of the eighties.

Information Engineering, often thrown into conversations for buzz-word compliance, is merely a disciplined approach to dynamic requirements gathering and software engineering. Utilizing Joint Application Development and RAD techniques it documents Business Process flow; its end product is the Data Model, the working blue print of what will be in the database.

Client/Server is more than the death of "big iron" mainframes -- the model where 'super computers' do all the thinking and are connected to dumb terminals which passively sit, acting as mere extensions of the mainframes' keyboards and monitors. Client/Server has, in fact, delivered much more, empowering IS groups with heretofore unapproachable degrees of flexibility – a new era for open systems. Suddenly one could mix-n-match the best-of-breed for each component in their system at almost any level. Everything in client/server is sold a la carte.

Client/Server provides Service, Shared Resources, Transparency of Location, Mix and Match, Message-based Exchanges, Encapsulation of Services, Scalability, Integrity. It's simultaneously downsizing, upsizing, and right-sizing.

Relational Database Management Systems (RDBMS) are built upon the considerable learning garnered by decades of advances with traditional databases and, further, are bolstered with the application of the significant research in set theory and predicate calculus. A modern RDBMS services requests for data from multiple clients at the same time, while providing an environment that protects the data against a variety of possible internal and external threats. It manages the recovery, concurrency, security, and consistency of a database. It frees system developers from the issues of efficient data storage and control.

Structured Query Language (always written as SQL and pronounced "sequel") is the standard language for interacting with all RDBMS--be it creating the tables, filling them with data, or administering users' access, SQL is the language spoken by all RDBMS regardless of database vendor.

Evolution of Databases

Relational databases help run the modern world - they are in phone switches, customer service systems, and electric utilities' power management grids, and are behind most of the interactive content that is available on the World Wide Web.

So the obvious questions are "how did they come about?" and "what is so unique about them?". To answer these questions flashback to the computer world of the late sixties, to around the early 1970's when choices for data storage were :

- Flatfiles
- Hierarchical databases (IBM IMS, HP IMAGE)

The limitations of flatfiles are fairly obvious – the most obvious being the need to sequentially access (top to bottom search) every record to perform any search. One way around this is to intelligently order the file, but then additions and deletions become quite painful. Files are really bad for lots of data.

An example of a hierarchy is "All the world in Parent -> Child". The problem is that not everything fits so neatly into this model. Problem: How to put employees into Hierarchy? Departments -> Employees -> phones

Network model -> Child can have multiple parents (marginally better, but still required complex navigation)

RDBMS

The story of relational database theory begins in 1969 when Dr. Edgar (Ted) F. Codd working as an IBM fellow, wrote an odd paper titled "A Relational Model for Large Shared Data Banks for Association for Computing Machinery (ACM)."

RDBMS is a "database of relations (two dimension array; flatfile) that has capability to recombine (join) data items to form different relations."

Thus the relational database model was created, which was nearly universally seen as a superior concept, but nonetheless took 12-15 years (1984) before a viable RDBMS were introduced. Why? Because of the requirements of the relational model relational databases required hefty hardware power and significant software engineering to pull it all together (the company that would eventually become Oracle arrived on the scene first in '79, still nearly a decade after Codd published). Some of the significant challenges that had to be overcome are listed here.

Relational Databases accomplish table joins through data in columns. Prior databases had embedded links or pointers between nodes. Pointers contained an absolute address of a structure. Fast access but very inflexible. Language limited to statements like GET NEXT WITHIN PARENT.

RDBMS have no embedded pointers. Deduce at execution time how to return the result. Simple select like `select name, address from customer where town="LA"` requires 30,000 executed instructions. Good news is that cpu's and the data access technology of computers now provide the power needed. Result is big wins in number of areas.

RDBMS make great use of the Client/Server, Requests-Response, model. Queries are made in SQL, database returns results. Very efficient use of network bandwidth.

You may have already heard of **Object-Relational Databases** and be wondering what these are. In fact, they are just an extension of RDMS. There are some on the market today which, in fact, have merely imposed an object oriented abstraction layer atop an RDBMS. A true, viable Object db isn't commercially ready, yet.

SQL Basics

Procedural versus Declarative

Procedural coding (3GL) contains statements that tell the computer exactly what to do. In contrast, a declarative language like SQL tells the computer what you want to achieve, and the computer decides how to affect the correct result.

Typically, the SQL statement required to manipulate a set of data will be much smaller than similar statements written in a third generation programming language such as COBOL or C. For example, the pseudo-code statements necessary to read through a table of data and list records which have a code of 'A' would be something like:

```
Open File
If file opened correctly
    Read a record
    While not EOF
        If code = 'A'
        Then
            List the record
        Read a record
    EndWhile
    Close File
Endif
```

An SQL statement to perform the same actions might be:

```
Select *
From Data
Where code = 'A'
```

As you can see, the SQL version is simpler, as is often the case. Additional benefits of the SQL approach are:

- No explicit record locking statements are needed, the server manages arbitration between users of data.
- No references to the physical locations of data, the server presents the user with a logical, tabular view of data.
- No indication of how to find the data, the server determines the most efficient method for accessing data

Sets

A set is an unordered collection of items, all of the same type and structure. In SQL these sets are called tables, and their elements are called **rows**. Rows are made up of **columns**.

People love to think of databases as fancy spreadsheets, partially because the interface to many databases does look like a spreadsheet. Both may be laid out in a grid, but that's about as far as the analogy holds.

Database is a series of Relations on Tables.

A Table is defined as rows and columns.

Column = field

Row = record.

Database -> Set of Tables

Table -> Described by its columns (fields)

Row -> is one record in a table

Consider the table of Employees (each Employee has Name, PhoneNumber, Dept)

Employees		
Name	PhoneExtension	Department
Sam Walton	555-8768 x878	Sales
Terry Inquino	876-5353	Marketing
Daniel Bell	989-5432	Sales
William Gibson	881-9879	Sales
Hart Crane	535-6226	Marketing
Grant Naylor	674-8768 x979	Accounting
David Shepard	987-7657 x314	Accounting
Merk Weinstein	555-7651 x8888	Accounting

Really cool thing about the relational model is that you can apply relational algebra to join tables together and the result is always: Another table.

Relationship Cardinality

A----B (One to One)

A ---< B (One to Many)

A>--<B (Many to Many) - Cannot be implemented in an RDBMS. Requires an intersection table to be created to resolve this relationship.

Tables versus Files

Tables are more conceptual and less physical than files. A file has a known ordering (that is there is a first record, a following record, a prior record, and a last record), a table has no ordering.

Tables, unlike files, may be joined using well understood set theory:

1. Joins occur via data. No hard links, so new data can be added and joined anytime it makes sense. Database becomes extensible.
2. Interactive querying - sit at a command line and ask questions - db will eventually compute answer. Not possible with files. To do the same with files actually coding, and a lot of it, is required.
3. SQL is a universal syntax. One db, many frontends. Being standard and non procedural means that it is more open; the frontend (user interface) and backend (database design) can be modified independently. Good and bad, but mostly good.

You can reap great rewards from using a standard, and SQL, set theory, and relational models are standard. This means any analyst familiar with these may step in and in a short period be productive using SQL rather than tedious and less universal procedural code. Tables are simple to understand whereas file-based storage is implementation specific.

Relational Algebra Operations:

Unary (One operand)

Selection - rows from table that meet a criteria (simple where clause)

Projection - Rows that contain some of the columns from the source table

- Binary (two operands)
- Union - rows from two tables are combined, removing dupes
- Intersection - rows common to two tables
- Difference - rows in first table but not in second
- Product - concatenation of all rows in one table with all rows in another
- Join - concatenation of rows from one relation to related rows from another

Transparent client access/ client platform independence: As long as the client knows how to request data from the server via the client library, you can use the RDBMS. As an example of this, the RedOrb server is storing user and login data in Oracle by making C language calls to the Oracle client library. It has no idea how data is stored or retrieved and doesn't need to know. It makes SQL calls and gets result sets back.

Another major difference between files and tables is a concept of the virtual tables, known as Views: Statserv page RedOrb report supported from view which hides details of the actual table design.

Yet another is that the DB determines the most efficient manner to answer a request, not the lone programmer find a handles. In fact, RDBMS have query optimization engines. These are:

- Rule Based- Syntax is parsed in a simple way and data is accessed predictably according to rules
- Cost Based- database computes the cost of various access methods, comes up with the best. Uses statistics about the nature of the data.

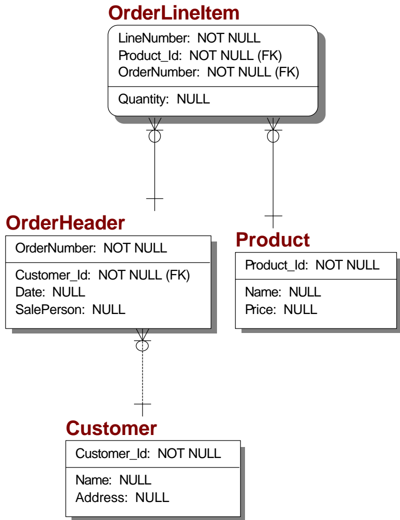
Big database problems: Parallel query capabilities.

Transaction Control

SQL supports transactions. A transaction is either completely finished or not done at all; there's no way to do only part of the work. This is the principle of working with sets. The work can be **committed**, which means it's made a permanent part of the database, or it can be **rolled back**, which means the database is restored to the state it was before the transaction began.

The figure at below is a **Data Model**, a visual representation of the tables within a database. This methodology (standard

for drawing these pictures) uses boxes to represent tables; the table name appearing above the box and the columns that make up the table appearing inside the box. This model also delineates Primary Keys by listing them at the top of the list column inside the boxes and then separating them from the other columns with a line.



Consider how important transaction support, the idea of a logical unit of work, is to an order entry system. We merely want to insert a new order into the system. Wee need to insert into has two tables: OrderHeader, which contains the CustomerId, and OrderLineItem, which has one line for each item in order. The Customer, name=Marty McFly, orders two items, one dozen Staplers and one gross of Staple. We then:

1. Create (insert) Order Header.

2. Insert Line Item 1, the product is Stapler.

3. Insert Line Item for the second product, Staples.

Now lets say that we got an (Error) on the insert of the second line item, Staples (lets assume this was because it was out of the stock and Marty wants it shipped today)! Without transaction support, the database is left in inconsistent state. Looks like McFly ordered one product, Stapler, when he really ordered two.

We are in luck, however. RDBMS provides Begin Transaction / End Transaction / Commit or Rollback. We can rollback the entire transaction and the order, would not be entered into the system. The user would have to ask Marty what to do.

Another aspect of transactions negotiated the many-horned problem of many people accessing the same data simultaneously. Multi-user locking protects corruption of record. Can offer additional levels of locking support. Depends on the application.

Deadlock (Deadly embrace) protection:

User A looks up Department = MIS wants to update employees with new department name of IS. User B looks up Courts Carter wants to change name of department that Courts is in to BIGMIS. How can both of these be done without someone's data being lost?

Lastly, the RDBMS provides Read vs. write consistency - someone running a summary report won't have to wait on a data entry person and vice versa. Called versioning. Oracle has it.

Keys

A requirement of relational databases is that every table have what is called a "Primary Key". The relational model assumes that each piece of the data it stores for you has some uniqueness, some special meaning that differentiates one row in a table from another. It, therefore, requires that for every row, in every table there must be some unique way of differentiation. After all, if all the rows in a table look alike what value or meaning does any one of them contain?

So in the OrderLineItem example given earlier we see that OrderNumber, Customer_Id, and Product_Id are the Primary Keys for Order, Customer, and Product, respectively.

We know that an Order must have be placed by a Customer and the model shows this via the dashed line connecting the Order table box and the Customer table box. The column Customer_No appears in the Order table. This is as we would expect, after all. Within the Order table the columns Customer_No is referred to as a Foreign Key – it is the link between an Order and the Customer who placed the Order.

Keys are talked about a lot in the relational world. In fact, it is the accounting and enforcement of clean keys to which the word relational in RDBMS refers – relations between tables via their keys. Codd saw this a major advantage over other database models. In an RDBMS the database strictly protects the integrity or validity of keys. The mantra of all RDBMS is:

> *It's the Key,*
> *the whole Key,*
> *and nothing but the Key*
> *so help me Codd.*

Simply defined a key is a column or set of columns in a table which can identify each row within that table. While a table might contain many possible key columns it is standard practice to select the most compact and stable column(s) as a table's Primary Key. There are many key descriptors:

- Surrogate Key… aka Technical Key (Psuedo/surrogate/(often sequential))

  Common practice to avoid putting information into a key. Usually Primary keys end up being surrogate keys in modern Entity - Relationship (ER) models.

- Intelligent Key… has imbedded meaning to users, usually a poor choice for a Key.
- Primary Key… guaranteed unique, Extant (immediately assigned)
- Alternate Keys … technically anything that isn't the primary key, often this just means other indexes
- Candidate Keys… could be a primary key/ can't be NULL.
- Foreign Key… is an attribute that is also the Primary Key of another table.
- Compound Key… aka MultiValue, Composite, Multi-Segmented, or Concatenated Key

(LastName+FirstName+MiddleInitial) - This wouldn't really be a good primary key would it? Why? (How many John Smiths are there?)

- Simple keys… based on a single attribute (PhoneNumber, SS#)

Many RDBMS have a built-in mechanism for unique key generation, an issue of great consternation for programmers in multi-user environments. Oracle has "sequences" which handle this for you :

```
INSERT INTO Customer (CustomerId) VALUES
(CustomerSeq.NextVal)
```

Integrity

Integrity constraints are rules that are part of the tables in a database. When you finish doing a transaction against the database, all these rules must be true.

Referential Integrity is the verification that links between table rows are valid at all times. These links are the physical implementation of the relationships modeled in the Entity-Relationship diagram during the logical design stage.

There are other types of integrity, one of which is the under appreciated constraint. Declarative integrity constraints and domains - keep data pure.

Example 1:

$0 < Age < 140$

Example 2:

State in Address will only allow pre-defined State Codes (i.e. "AR","CA","NY"). There's no way to insert a state of "XX".

- Domain… Adds meaning to an attribute. Physical and logical domains exist for each attribute in a table.
- Physical Domain… identifies the format (field data type + size)
- Logical Domain… specifies the set of allowable values.

All of this ensures data consistency - rules are enforced where and when the data is stored and there is no way to circumvent the rules and corrupt data.

Normalization

Ensures that each fact in the db appears only once, and that they are arranged in a way that does not lead to ambiguous meanings. Much more on this later.

Stored Procedures

These were an innovation of Sybase, and in 1986 when they were first introduced, they turned the industry on its ear. I still think they rock. Essentially a stored procedure is a set of SQL commands that are stored on the server. They move business logic and functionality to the server-side and drastically lower the amount of network traffic common with ad hoc or dynamic SQL statements. As if that is not reason enough to jump on the bandwagon these things are fast. Since they reside in the server the execution plan is precompiled – the db doesn't have to think about the best way to get the results. After the first time a stored procedure is executed it doesn't get reevaluated (unless forced). Another bene is that one can create a virtual db api that hides complex database designs. Lastly, security. A stored procedure can do things that are prohibited otherwise. They allow you to create an api that allows a user to update a series of tables without giving explicit grants on these tables to the individual.

Triggers

Triggers are special user-defined actions that are automatically invoked by the RDBMS based on data-related events. An event tells you something happened to the database; a trigger is an event handler written to take the proper action in response to that event.

In the case of Oracle there are 12 triggers per table which are supported. It does that by allowing you to specify for each Insert/Update/Delete the following: a before trigger that fires before the statement executes, and an after trigger that fires after the statement executes. In addition, Oracle lets you specify the number of times a trigger fires. Row-level triggers fire once for each updated row; statement-level triggers fire once for the entire SQL statement, even if no rows are inserted, updated, or deleted. Both can be defined to be active simultaneously. Lastly, more than one trigger per operation can be defined, but the order of these is not controlled or known by you!