GUI Application For Generating SQL Using Venn Diagram (Concept & Algorithms)

Anand Jani
Maintanence and Technical Support
Pacttra Inc. Shanghai, China.

Received July 02, 2014. Accepted August 15, 2014.

ABSTRACT SQL provides the facility to accessing and manipulating databases through query. But writing a query becomes too complex if there are too many numbers of tables. Graphical representation has been proved an easy way of understanding any things. Generating a report is a strong corner of any application but triggering a query can be an issue for a user. It is an obvious that we can create a Venn diagram for a joined query. So its reverse is also possible i.e. to create a joined query from a Venn diagram. Set theory is a basic mathematics concepts and can be easily understand and generated by any user rather than learning SQL. The proposed application provides to generate SQL statement through Venn diagrams. There will be a parser which will generate a query from it and display the result set. Query would be equivalent to DMLs but no need to write and it becomes easy to get an appropriate output when there are too many numbers of tables. This application can be integrated with any information systems to generate reports.

Key words **:** Generating SQL

Introduction

Remembering my old class days, I remember the time when I have so many doubts about the JOIN operations in SQL query. Then my teachers used to show me the Venn diagrams to make it simpler. Then I truly understood how the visualization is powerful in making things understandable. How two tables are joined together and how the common records are fetched.

Finally we concluded that it is possible to create a Venn diagram for any joined query. Later we came up with an idea that if it is possible to create Venn Diagrams from a query then the reverse process should also be possible. That is to create an SQL query from the prepared Venn Diagram. Where one circles (Set) shows one relation from the database.

SQL is a traditional way of manipulating the database. All the database application uses the same technique for the records from the database. But the problems with the SQL query are it is syntactically dependant and bias to English language. It also becomes too complex when there are too many joins in a single query. Graphics have been proved an easy way to deal with the complexity in all domains. So we can have an application that lets users to generate the diagrams, performs operations and then giving SQL query that can retrieve the data from the RDBMS.

Advantages of such application would be…

- They are less syntactically demanding for SQL.
- Overcome the language dependency.
- Can be integrated with any Information System to generate reports.

Set theory is a basic mathematics concepts and can be easily understand and generated by any user rather than learning SQL. Venn diagrams was introduced in 1883 by John are diagrams that show all possible logical relations between a finite collection of sets.

Textual query languages based on Boolean logic are common amongst the search facilities of on-line information repositories. However, there is evidence to suggest that the syntactic and semantic demands of such languages lead to user errors and adversely affect the time that it takes users to form queries. Additionally, users are faced with user interfaces to these repositories which are unresponsive and uninformative, and consequently fail to support effective query refinement. We suggest that graphical query languages, particularly Venn-like diagrams, provide a natural medium for Boolean query specification which overcomes the problems of textual query languages. Also, dynamic result previews can be seamlessly integrated with graphical query specification to

increase the effectiveness of query refinements.

A direct-manipulation user interface which exploits diagrammatic techniques for query specification can circumvent problems with textual query languages.

Graphical query notations can have two key advantages over textual query languages:

1.) They are less syntactically demanding.

2.) Overcome the English language and Boolean operator ambiguity.

A huge number of potential join edges between pairs of tables, making it particularly challenging to understand their structure and to formulate non-trivial join queries. Business data analysts are often faced with exactly this challenging task, since they need to pose complex ad hoc join queries to perform sophisticated analyses of a variety of enterprise-wide processes whose data is buried in these databases. Graphical tools for query formulation are helpful when the analysts already have a comprehensive understanding of the schemas and the tables that need to be joined for the analysis task at hand. When the analyst does not have such a thorough understanding of the database, formulating such complex SQL join queries becomes tedious, especially over databases with large, heterogeneous schemas.

When the desired query is a complex join query, the SQL query log is unlikely to contain any query whose sets of start and end tables exactly match the user specification.In existing system, SQL provides the facility to accessing and manipulating databases through query. But writing a query becomes too complex if there are too many numbers of tables. So for that solution, Graphical representation has been proved an easy way of understanding any things. Generating a report is a strong corner of any application but triggering a query can be an issue for a user. It is an obvious that we can create a Venn diagram for a joined query.

An alternate method for determining the validity of categorical syllogisms is the Venn diagram method. The conventions of this method are,

1.) to represent categorical claims with interlocking circles;

2.) each circle represents a term;

3.) an asterisk indicates that at least one thing exists in the area where it is placed;

4.) stroking out an area indicates that there is nothing in that area.

Venn diagrams are very useful for visualizing the relationships between groups.In order to use these diagrams to test for validity we must link three circles together. A categorical syllogism has three terms and since each circle represents one term, three circles will be needed. Any of the three propositions in the syllogism can be diagrammed by using the two circles which represent that proposition's terms and (in some ways) ignoring the third circle.

To test for validity one diagrams **only the two premises**. Then one looks at the diagram to see whether anything would need to be added to diagram the conclusion. Since the conclusions of valid arguments do not claim more than the information given in their premises, if more would have to be added to diagram the conclusion, that conclusion must claim more than the information given in the premises and hence be invalid. In the event of an invalid argument one leaves the conclusion undiagrammed to demonstrate the invalidity of the argument.

Another problem is that what a DML ignores is not easily visible, mainly because we do not have criteria telling us what a DML should include. This implies categorization of DML characteristics, which should ideally be dependent upon usage context (domain, task) and need. Such categorizations would also be instrumental in evaluating/choosing DMLs for different tasks.

Data Step Graphics Interface (DSGI) was used to draw the Venn diagrams. The GSET function was used to set the Line Color and the Line Width of the Venn Diagram Ellipse. We measured the time to generate a join query recommendation in response to a user provided specification. Our graphical methods are very fast and the average response time. Response time includes the accuracy and confidence measures and writing the results into files.

Venn diagram

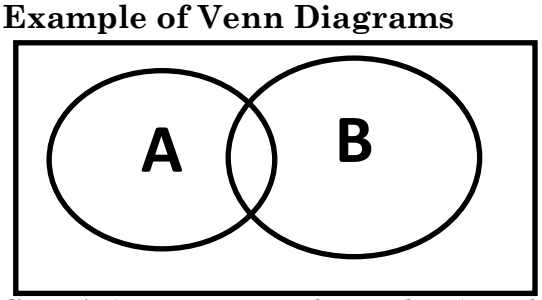
A Venn diagram is constructed with a collection of simple closed curves drawn in a plane. According to Lewis (1918), the "principle of these diagrams is that classes be represented by regions in such relation to one another that all the possible logical relations of these classes can be indicated in the same diagram. That is, the diagram initially leaves room for any possible relation of the classes, and the actual or given relation, can then be specified by indicating that some particular region is null or is not-null".

Venn diagrams normally comprise overlapping circles. The interior of the circle symbolically represents the elements of the set, while the exterior represents elements that are not members of the set. For instance, in a two-set Venn diagram, one circle may represent the group of all wooden objects, while another circle may represent the set of all tables. The overlapping area or *intersection* would then represent the set of all wooden tables. Shapes other than circles can be employed as shown below by Venn's own higher set diagrams. Venn diagrams do not generally contain information on the relative or absolute sizes (cardinality) of sets; i.e. they are schematic diagrams.

Venn diagrams are similar to Euler diagrams. However, a Venn diagram for n component sets must contain all 2^n hypothetically possible zones that correspond to some combination of inclusion or exclusion in each of the component sets. Euler diagrams

contain only the actually possible zones in a given context. In Venn diagrams, a shaded zone may represent an empty zone, whereas in an Euler diagram the corresponding zone is missing from the diagram. For example, if one set represents *dairy products* and another *cheeses*, the Venn diagram contains a zone for cheeses that are not dairy products. Assuming that in the context *cheese* means some type of dairy product, the Euler diagram has the cheese zone entirely contained within the dairy-product zone—there is no zone for (non-existent) non-dairy cheese. This means that as the number of contours increase, Euler diagrams are typically less visually complex than the equivalent Venn diagram, particularly if the number of non-empty intersections is small.

Example of Venn Diagrams



Sets A (creatures with two legs) and B (creatures that can fly)

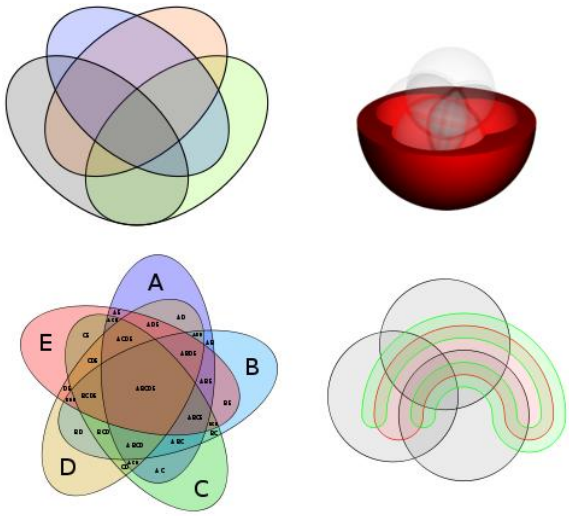
The following example involves two sets, A and B, represented here as colored circles. The orange circle set A, represents all living creatures that are two-legged. The blue circle, set B, represents the living creatures that can fly. Each separate type of creature can be imagined as a point somewhere in the diagram. Living creatures that both can fly *and* have two legs—for example, parrots—are then in both sets, so they correspond to points in the area where the blue and orange circles overlap. That area contains all such and only such living creatures.

Humans and penguins are bipedal, and so are then in the orange circle, but since they cannot fly they appear in the left part of the orange circle, where it does not overlap with the blue circle. Mosquitoes have six legs, and fly, so the point for mosquitoes is in the part of the blue circle that does not overlap with the orange one. Creatures that are not two-legged and cannot fly (for example, whales and spiders) would all be represented by points outside both circles.

The combined area of sets A and B is called the union of A and B, denoted by A ∪ B. The union in this case contains all living creatures that are either two-legged or that can fly (or both). The area in both A and B, where the two sets overlap, is called the intersection of A and B, denoted by A ∩ B. For example, the intersection of the two sets is not empty, because there *are* points that represent creatures that are in *both* the orange and blue circles.

Extensions to higher numbers of sets

Venn diagrams typically support two or three sets, but there are forms that allow for higher numbers. Shown below, four intersecting spheres form the highest order Venn diagram that is completely symmetric and can be visually represented. The 16 intersections correspond to the vertices of a intersect (or the cells of a 16-cell respectively).



SQL and Venn Diagram's relation

Now lets discuss an example. I am going to discuss seven different ways you can return data from two relational tables. I will be excluding cross Joins and self referencing Joins. The seven Joins I will discuss are shown below …

Suppose we have two tables, *Table_A* and *Table_B*. The data in these tables are shown below:

TABLE_A

PK	Value
1	FOX
2	COP
3	TAXI
6	WASHINGTON
7	DELL
5	ARIZONA
4	LINCOLN
10	LUCENT

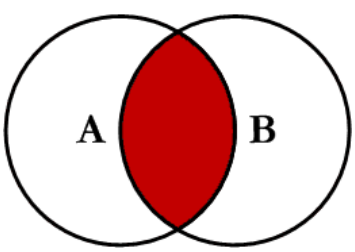
TABLE_B

PK	Value
1	TROT
2	CAR
3	CAB
6	MONUMENT
7	PC
8	MICROSOFT
9	APPLE
11	SCOTCH

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. OUTER JOIN
5. LEFT JOIN EXCLUDING INNER JOIN
6. RIGHT JOIN EXCLUDING INNER JOIN
7. OUTER JOIN EXCLUDING INNER JOIN

For the sake of this article, I'll refer to 5, 6, and 7 as LEFT EXCLUDING JOIN, RIGHT EXCLUDING JOIN, and OUTER EXCLUDING JOIN, respectively. Some may argue that 5, 6, and 7 are not really joining the two tables, but for simplicity, I will still refer to these as Joins because you use a SQL Join in each of these queries (but exclude some records with a WHERE clause).

Inner JOIN

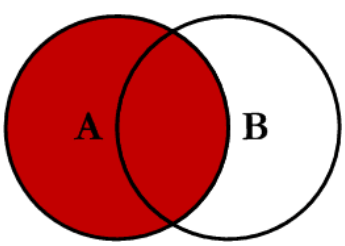


This is the simplest, most understood Join and is the most common. This query will return all of the records in the left table (table A) that have a matching record in the right table (table B). This Join is written as follows:

```
SELECT <select_list>
FROM Table_A A
INNER JOIN Table_B B
ON A.Key = B.Key
```

```
-- INNER JOIN
SELECT A.PK AS A_PK, A.Value AS
A_Value,
B.Value AS B_Value, B.PK AS B_PK
FROM Table_A A
INNER JOIN Table_B B
ON A.PK = B.PK

A_PK A_Value    B_Value    B_PK
---- ---------- ---------- ----
1 FOX        TROT          1
2 COP        CAR           2
3 TAXI       CAB           3
6 WASHINGTON MONUMENT      6
7 DELL       PC            7

(5 row(s) affected)
```

Left JOIN



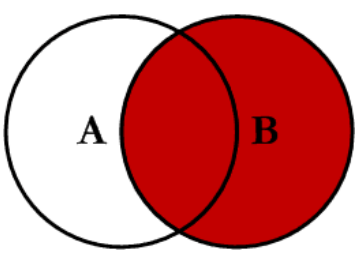
This query will return all of the records in the left table (table A) regardless if any of those records have a match in the right table (table B). It will also return any matching records from the right table. This Join is written as follows:

```
SELECT <select_list>
FROM Table_A A
LEFT JOIN Table_B B
ON A.Key = B.Key
```

```
-- LEFT JOIN
SELECT A.PK AS A_PK, A.Value AS
A_Value,
B.Value AS B_Value, B.PK AS B_PK
FROM Table_A A
LEFT JOIN Table_B B
ON A.PK = B.PK

A_PK A_Value    B_Value    B_PK
---- ---------- ---------- ----
   1 FOX        TROT          1
   2 COP        CAR           2
   3 TAXI       CAB           3
   4 LINCOLN    NULL       NULL
   5 ARIZONA    NULL       NULL
   6 WASHINGTON MONUMENT      6
   7 DELL       PC            7
  10 LUCENT     NULL       NULL

(8 row(s) affected)
```

```
-- RIGHT JOIN
SELECT A.PK AS A_PK, A.Value AS
A_Value,
B.Value AS B_Value, B.PK AS B_PK
FROM Table_A A
RIGHT JOIN Table_B B
ON A.PK = B.PK

A_PK A_Value    B_Value    B_PK
---- ---------- ---------- ----
   1 FOX        TROT          1
   2 COP        CAR           2
   3 TAXI       CAB           3
   6 WASHINGTON MONUMENT      6
   7 DELL       PC            7
NULL NULL       MICROSOFT     8
NULL NULL       APPLE         9
NULL NULL       SCOTCH       11

(8 row(s) affected)
```

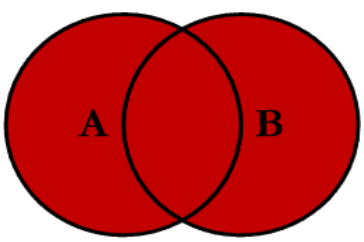
Right JOIN



This query will return all of the records in the right table (table B) regardless if any of those records have a match in the left table (table A). It will also return any matching records from the left table. This Join is written as follows:

```
SELECT <select_list>
FROM Table_A A
RIGHT JOIN Table_B B
ON A.Key = B.Key
```

Outer JOIN



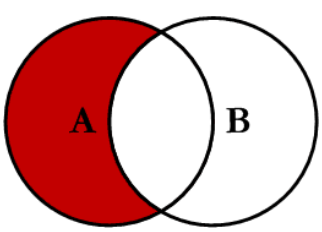
This Join can also be referred to as a FULL OUTER JOIN or a FULL JOIN. This query will return all of the records from both tables, joining records from the left table (table A) that match records from the right table (table B). This Join is written as follows:

```
SELECT <select_list>
FROM Table_A A
FULL OUTER JOIN Table_B B
ON A.Key = B.Key
```

```
-- OUTER JOIN
SELECT A.PK AS A_PK, A.Value AS
A_Value,
B.Value AS B_Value, B.PK AS
B_PK
FROM Table_A A
FULL OUTER JOIN Table_B B
ON A.PK = B.PK

A_PK A_Value    B_Value    B_PK
---- ---------- ---------- ----
   1 FOX        TROT          1
   2 COP        CAR           2
   3 TAXI       CAB           3
   6 WASHINGTON MONUMENT      6
   7 DELL       PC            7
NULL NULL       MICROSOFT     8
NULL NULL       APPLE         9
NULL NULL       SCOTCH       11
   5 ARIZONA    NULL       NULL
```

```
-- LEFT EXCLUDING JOIN
SELECT A.PK AS A_PK, A.Value AS
A_Value,
B.Value AS B_Value, B.PK AS B_PK
FROM Table_A A
LEFT JOIN Table_B B
ON A.PK = B.PK
WHERE B.PK IS NULL

A_PK A_Value    B_Value    B_PK
---- ---------- ---------- ----
   4 LINCOLN    NULL       NULL
   5 ARIZONA    NULL       NULL
  10 LUCENT     NULL       NULL
(3 row(s) affected)
```

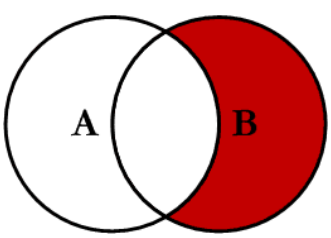
Left Excluding JOIN



This query will return all of the records in the left table (table A) that do not match any records in the right table (table B). This Join is written as follows:

```
SELECT <select_list>
FROM Table_A A
LEFT JOIN Table_B B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

Right Excluding JOIN



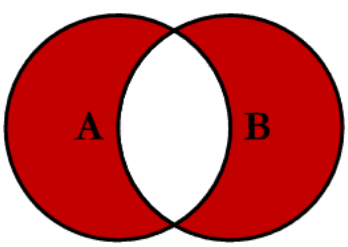
This query will return all of the records in the right table (table B) that do not match any records in the left table (table A). This Join is written as follows:

```
SELECT <select_list>
FROM Table_A A
RIGHT JOIN Table_B B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
-- RIGHT EXCLUDING JOIN
SELECT A.PK AS A_PK, A.Value AS
A_Value,
B.Value AS B_Value, B.PK AS B_PK
FROM Table_A A
RIGHT JOIN Table_B B
ON A.PK = B.PK
WHERE A.PK IS NULL

A_PK A_Value    B_Value    B_PK
---- ---------- ---------- ----
NULL NULL       MICROSOFT     8
NULL NULL       APPLE         9
NULL NULL       SCOTCH       11

(3 row(s) affected)
```

```
-- OUTER EXCLUDING JOIN
SELECT A.PK AS A_PK, A.Value AS
A_Value,
B.Value AS B_Value, B.PK AS B_PK
FROM Table_A A
FULL OUTER JOIN Table_B B
ON A.PK = B.PK
WHERE A.PK IS NULL
OR B.PK IS NULL

A_PK A_Value    B_Value    B_PK
---- ---------- ---------- ----
NULL NULL       MICROSOFT     8
NULL NULL       APPLE         9
NULL NULL       SCOTCH       11
   5 ARIZONA    NULL       NULL
   4 LINCOLN    NULL       NULL
  10 LUCENT     NULL       NULL

(6 row(s) affected
```

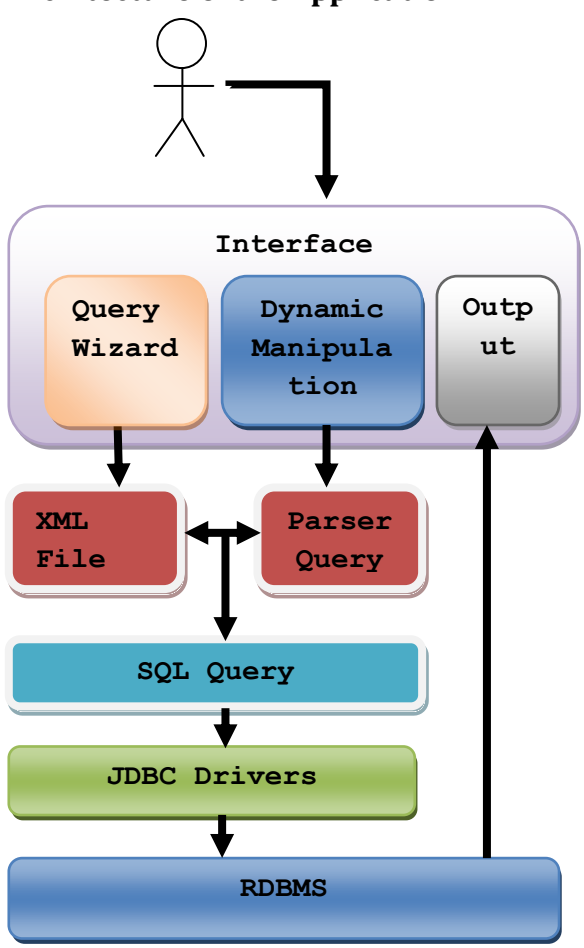
Outer Excluding JOIN



This query will return all of the records in the left table (table A) and all of the records in the right table (table B) that do not match. I have yet to have a need for using this type of Join, but all of the others, I use quite frequently. This Join is written as follows:

SELECT *<select_list>*
FROM Table_A A
FULL OUTER JOIN Table_B B
ON A.Key = B.Key
WHERE A.Key IS NULL OR B.Key IS NULL

Architecture of the Application



User interacts with the GUI interface to decide the initial parameters like database name, table name, attribute selection for the table. This is generally done with the Query wizard which is a menu. Interaction with the menu creates an XML file for each table and acts as metadata for the future use. The XML file contains the information about the table like name, primary key, attributes to display, attribute selection criteria, selection conditions, selection key to specifies the joining tables, different clause information and its criteria and etc.. There is a one separate file is managed for one single table and it will be used by the Parse to join the Tables. The joing process is achieved dynamically as the user interacts with the Set (Circles). The interaction is in the term of Drag-Drop operations to join the table. The Parser dynamically at the run time join the table reading the Venn Diagrams.

The Parser build the SQL Query using the XML File in the consequences of the dynamic manipulation of the Venn diagram thought the interface. The Parser works in two different phases. Phase 1 identifies the tables to join and second phase append the query.

The Application establishes connection with the actual RDBMS with the help of JDBC Driver. RDBMS retrieve the result set from the database and return back to the application interface. The result obtained can also be treated as another Set and can be manipulated further.

Query Structure

If we want any application to build a Query then first of all we should know the structure of a query. I would divide a single query (not including inner query but may include a join) in four different parts.

Query = Projection + Tables + Conditions + Clauses

Projection is the attributes selection. Tables are the tables names from the records are to be displayed. Conditions can be of two types.

1. Conditions that decides the join criteria.
2. Conditions that decides the projection criteria.

The only decision to prepare the conditions that decides the join criteria is complex one as it consists of

the dynamic decisions according to the Graphic object of a set.

Application Introduction

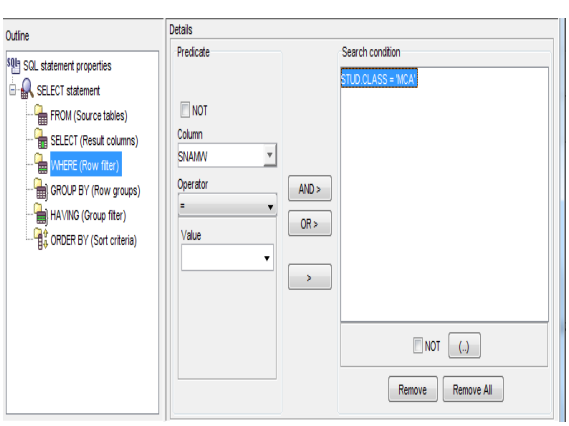
It can be any GUI application that let a user initially select the tables, attributes, projection criteria. Then application will display a Venn Diagram for the selected relation with the specified attributes. User can create Venn Diagrams for more than one relation. As a circle on screen is a moving dynamic object it can be dragged and dropped. As User drag and drop a circle on the other, it will generate the join for that two tables as they are now intersecting with each other. The same process may be carried out with N numbers of circles. Clicking the middle area intersection will display the joined record of both the relation.

Following can be the Steps to generate a Venn Diagram for a relation.

1. Select database.
2. Select Table.
3. Select Projection i.e. Attribute to display (Primary key is included by default to keep user away from the complexity)
4. Select condition.
5. Select AND clause (Can be repetitive)
6. Select OR clause
7. Select Clause Group by, Order by & Having
8. Click "*Generate Diagram* "

First users selects the database and then interested table let say X. Then he adds the attributes for the projection let say X.a, X.b , X.c. Some default attributes are included by default like primary key as the Join operation in future will be requiring that. Then he applies the selection conditions for the projection. For example he selects X.b < 500 and rest of clause like group by, order by and having. When he is done with this wizard the finally presses the "Generate Diagram". And one circle will be appearing on the screen for relation X.

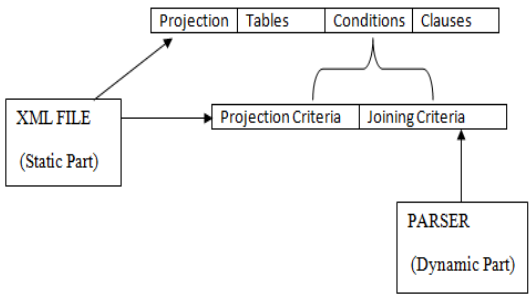
This wizard can be very much like IBM DB2's SQLAssists…



Let's get Technical

Rather than building a query directly it is better to divide it in different part and assigning it to different component. The hardest part is to identify the joining tables dynamically. As it has to be derived from the graphical object and which is dynamic too. So the from the query structure showed earlier part "condition that decides the joining criteria" is the difficult task. We have to develop a parser that read the Venn Diagrams and build that part and rest of part of query is generated by an XML file which is prepared by the table selection. So for the creating queries there are mainly two components.

1. XML file
2. Parser



XML FILE

XML can be used as metadata or data structure in modern computer science due to its variable length characteristics and flexibilities. The file format is under…

In the XML file, in <select >, <where> there can be more than one <column> tabs

```
<table id="table_id" name="table_name" alice="table_alice"
      primary-key="p1,p2" key-to-join="p1">
      <select>
            <column name="Col_name" As="Display_name">
            </column>
      </select>
      <where>
            <column-name name="Col_name" not="true/false" operator="opr"
            value="value" logical-operator="AND/OR">
            </column-name>
      </where>
      <group-by column-name="col_name" > </group-by>
      <having>
            <column-name="col_name" operator="opr" value="value">
      </having>
      <order-by>
            <column-name="col_name" order="ASC/DESC">
            </column-name>
      </order-by>
</table>
```

Parser

In computer science and linguistics, **parsing**, or, more formally, **syntactic analysis**, is the process of analyzing a text, made of a sequence of tokens (for example, words), to determine its grammatical structure with respect to a given (more or less) formal grammar. Parsing can also be used as a linguistic term, for instance when discussing how phrases are divided up in garden path sentences.

Parsing is also an earlier term for the diagramming of sentences of natural languages, and is still used for the diagramming of inflected languages, such as the Romance languages or Latin. The term parsing comes from Latin *pars* (*ōrātiōnis*), meaning part (of speech).

Parsing is a common term used in psycholinguistics when describing language comprehension. In this context, parsing refers to the way that human beings, rather than computers, analyze a sentence or phrase (in spoken language or text) "in terms of grammatical constituents, identifying the parts of speech, syntactic relations, etc." This term is especially common when discussing what linguistic cues help speakers to parse garden-path sentences. The *task* of the parser is essentially to determine if and how the input can be derived from the start symbol of the grammar. This can be done in essentially two ways:

- Top-down parsing- Top-down parsing can be viewed as an attempt to find left-most derivations of an input-stream by searching for parse trees using a top-down expansion of the given formal grammar rules. Tokens are consumed from left to right. Inclusive choice is used to accommodate ambiguity by expanding all alternative right-hand-sides of grammar rules.
- Bottom-up parsing - A parser can start with the input and attempt to rewrite it to the start symbol. Intuitively, the parser attempts to locate the most basic elements, then the elements containing these, and so on. LR parsers are examples of bottom-up parsers. Another term used for this type of parser is Shift-Reduce parsing.

LL parsers and recursive-descent parser are examples of top-down parsers which cannot accommodate left recursive productions. Although it has been believed that simple implementations of top-down parsing cannot accommodate direct and indirect left-recursion and may require exponential time and space complexity while parsing ambiguous context-free grammars, more sophisticated algorithms for top-down parsing have been created by Frost, Hafiz, and Callaghan which accommodate ambiguity and left recursion in polynomial time and which generate polynomial-size representations of the potentially-exponential number of parse trees. Their algorithm is able to produce both left-most and right-most derivations of an input with regard to a given CFG.

As show in above diagram, the Parser plays the most important role in building the SQL query as it generates the dynamically part of the query which defines the criteria for the Joing the tables. The functionality of our parser is to read the Venn Diagram, interpret and modify the query. The parser will have two passes.

Pass 1. Find the tables to join with each other.

Pass 2. Embed the SQL query with the join conditions.

Pass 1

In first pass a Matrix of M*N is created with initialized with the zero. Where M = N = Total numbers of the tables. The total numbers are retrieved from the XML file.

	Tbl1	Tbl2	Tbl3
Tbl1	0	0	0
Tbl2	0	0	0
Tbl3	0	0	0

Parser then scans the Venn diagram to find the information to join the tables that is which tables to join. This is done with the help of a function isIntersecting().

boolean isIntersecting(Rtbl,Ltbl)

Function *isIntersecting()*

Some specifications about the function are as under.

- It returns Boolean values.
- If two Venn diagrams are intersecting then returns true else false.
- The function logic

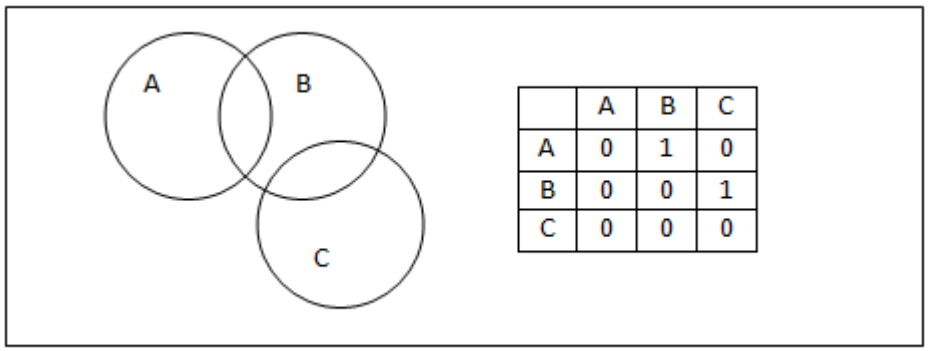
- As the Venn diagram is the moving object on the screen, the covered pixels by a single diagram may get changed. We can acquire all the covered pixels for both the diagrams. If there are any common pixels, it means the circles are intersecting. So the function returns TRUE.

- Code implementation depends on the technology we are going to use.

Switching back to the pass 1 logic, the following pseudo code may be used.

```
Set Rtbl = tbl1
WHILE tbl1 < = N
      Ltbl = Rtbl+1
      WHILE Ltbl <= N
            IF isIntersecting(Rtbl,Ltbl) EQUAL TRUE)
                  Matrix[Rtbl][Ltbl]=1
            ELSE
                  Ltbl =Ltbl + 1
      ENDWHILE
      Rtbl  = Rtbl + 1
ENDWHILE
Where,
      N = Total numbers of table
      Rtbl = Right table in join
      Ltbl = Left table in join
```

Pass 1 check for each table's corresponding Venn Diagrams and its intersecting diagrams and mark the corresponding cell with 1 in the Matrix. So finally a matrix is achieved which contains the information about joining the tables. Consider the following example…



	A	B	C
A	0	1	0
B	0	0	1
C	0	0	0

Pass 2

```
Set I=0
WHILE I < Total Column in Matrix
     Set J = 0
     WHILE J < Total Row in Matrix
          IF Matrix [I][J] EQUAL 1
                    ADD TO QUERY "tbl[I].ID = tbl[J]"
          ELSE
               J = J+1
     ENDWHILE
     I = I+1
ENDWHILE
```

Pass 2 starts reading the Matrix which was constructed by Pass 1. It starts reading each row and column one by one and checks if there for the value '1'. If it finds one in any cell, it means that the table name corresponding row and column have to be joined. So it adds the joining condition to the part in SQL query.

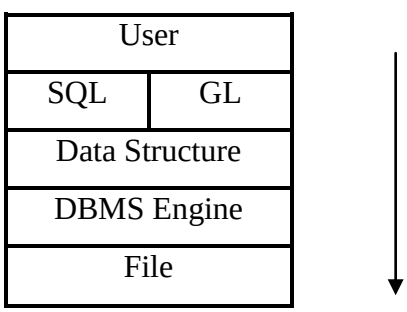
Conclusion

If we look at the process of retrieving the data from any DBMS, the process begins with the User; he manipulates the database using SQL. SQL

intends to pass the query to the database engine, but the query is not passed in its same format as user builds it. It if fist converted to proper data structure that engine can understand. So the overall process can be shown as under.

Implementing the idea of proposed application may increase the granularity easiness. But if look at another side of the coin, it increase the gap between user and the data, increasing the process overheads. As GL(Graphical Language) seats between user and SQL.

We don't claim that the proposed idea is perfect, issues and bugs are certain as if the more research are made regarding the topic or we elaborate it, So as the solutions too. But it opens a new door to the GML as a substitute to SQL. The main instructions to the engines are data structures. Researches should be made in the direction of GML that can generate data structure. This can replace SQL making process easier and simple.



A creative man is motivated by the desire to achieve, not by the desire to beat others.

~ Ayn Rand