An Extended Structured Query Language Incorporating Object Data Types for the Construction Industry

KEVIN C. HOLLISTER
CHARLES J. KIBERT
M.E. Rinker, Sr. School of Building Construction
University of Florida
Gainesville, Florida 32611 USA

ABSTRACT
The key to developing a system of information handling in construction that will have a dramatic positive impact on productivity is the development of an appropriate database language. This research involved the development of a standard construction industry specific extension to the Standard Query Language (SQL), the de facto international standard for relational database query languages. Construction Industry-SQL (CI-SQL), is the first industrial extension to SQL and serves as a prototype for a host of other extensions from other industries. CI-SQL uses object data, object attributes, and user-defined extensions to allow rapid access to industry databases. CI-SQL can play a significant role in the way information is retrieved by construction industry, allowing the development of robust databases that support SQL. Ultimately the implementation of CI-SQL will force industry to reconsider the heavy duplication of information that negatively affects productivity and will allow the introduction of keyless data entry and retrieval systems that enhance the speed and accuracy of information handling. Examples in construction material procurement will be used to illustrate the potential for CI-SQL applications. The ultimate use of CI-SQL will be to allow the use of keyless data entry systems, the foremost of which is bar coding, in construction specifications.

Key Words
SQL; object data; object attributes; keyless data entry; bar coding

Information Accessing Needs of Construction Industry

In few industries is accessing information more crucial than in the construction industry. Issues such as retrieving detailed information on the availability and cost of construction materials, as well as determining the interrelationships among project activities, are critical components in providing the construction industry with the information services it requires. Productivity increases can be realized through improved methods of information retrieval. These improvements are primarily derived from the addition of industry-specific vocabulary. With multiple key players (e.g., the owner, the architect, and the contractor) timely and accurate information sharing is a vital ingredient to a successful project. It is envisioned that massive increases in productivity can be attained by using bar codes in product specifications and product catalogs so that designers and constructors can query manufacturers' product data bases on-line via a Wide Area Network (WAN). The move from verbal requests for product data, availability, pricing, and ordering to keyless data entry and network information query is estimated to have a minimum factor of 10 potential increase in productivity in information handling. The development of a construction industry specific query language, as described in this paper, is the first step and the key to attaining this dramatic leap in productivity.

Origin of Relational Databases

In 1970, while working at IBM's Research Laboratory in San Jose, California, E.F. Codd published his definitive paper, "A Relational Model of Data for Large Shared Data Banks" (Codd, 1970). The technology for the entire domain of relational data bases emanated from this foundational work. In this paper, Codd "laid down a set of abstract principles for data base management: the so-called *relational model*" (Date, 1989). According to Pascal (1989), "His relational model, based on the set mathematics of relations of first-order predicate logic, covers the three aspects of data that any DBMS must address: structure, integrity, and manipulation".

Codd's Twelve Fidelity Rules

To counter misunderstandings and distortions, Codd later created his now famous Twelve Fidelity Rules, a minimum of six of which must be met in order for a data base management system to be considered truly relational. Although this set of rules is titled The Twelve Fidelity Rules, Codd intentionally began with Rule 0 which serves as a mandatory foundation for all relational DBMSs. These rules, as outlined by Pascal in "A Brave New World" (Pascal, 1989) are as follows:

Rule 0: Foundation Rule

Any system that is advertised as, or claimed to be, a relational DBMS, must manage the data base entirely through its relational capabilities.

Rule 1: Information Rule

All information in a relational data base must be represented explicitly at the logical level in exactly one way by table values.

Rule 2: Guaranteed Access Rule

Each and every data value in a relational data base is guaranteed to be logically accessible by resorting to a combination of table name, column name, and primary key value.

Rule 3: Missing Information Rule

Missing value indicators distinct from empty character strings, strings of blank characters, zero, or any other numbers, must represent and support, in operations at the logical level in a systematic way independent of data type, the fact that values are missing for applicable and inapplicable information.

Rule 4: System Catalog Rule

The description of the data base is represented at the logical level dynamically like ordinary data so that authorized users can apply the same (relational) language to its interrogation.

Rule 5: Comprehensive Language Rule

No matter how many languages and terminal interactive modes are supported, at least one language must be supported that is expressible as character strings per some well-defined syntax that supports interactively by program:

1 - data definition
2 - integrity constraints
3 - data manipulation
4 - views
5 - transaction boundaries
6 - authorization privileges

Rule 6: View Updatability Rule

The DBMS must have a way of determining at view definition time whether a view can be used to insert rows, delete rows, or update which columns of its underlying base tables and store the results in the system catalog.

Rule 7: Set Level Updates Rule

The capability of operating on whole tables applies not only to retrieval but also to insertion, modification, and deletion of data.

Rule 8: Physical Data Independence Rule

Application programs and interactive operations should not have to be modified whenever changes are made in internal storage or access methods.

Rule 9: Logical Data Independence Rule

Application programs and interactive operations should not have to be modified whenever certain types of changes involving no loss of information are made to the base tables.

Rule 10: Integrity Independence Rule

Application programs and interactive operations should not have to be modified whenever changes are made in integrity constraints defined by the data language and stored in the catalog.

Rule 11: Distribution Independence Rule

Application programs and interactive operations should not have to be modified whenever data is distributed or redistributed on different computers.

Rule 12: Nonsubversion Rule

If a DBMS has a low-level (procedural) language, that language should not be allowed to subvert or bypass integrity constraints or security constraints expressed in the high level relational level.

Principles of SQL

Codd's relational model sets forth both the precepts and the structure of relational data bases. A relational data base is characterized by its simplicity of data management, independence of logical user views from the physical data storage structure, and the availability of simple but powerful relational operators (Wipper, 1989). These characteristics translate into a collection of tables that are composed of rows and columns. Rows, also called tuples, contain the data associated with each of the tables' columns. Each column in a table is assigned a unique name and contains a particular type of data (Trimble & Chappell, 1989), such as an employee number for each employee in a personnel file. Acting conceptually as a storage medium, the intersections of rows and columns, sometimes called fields, provide the vehicle necessary for locating and accessing relational data. The number of rows in a given table is referred to as the cardinality of that table. The number of columns is called the degree (Date, 1989). These rows and columns form two types of tables. The first type, a base table, may be thought of as a table that actually exists; whereas a viewed table, or view, is a virtual table that is extracted from a single base table or a combination of base tables. Another relational data base component, the catalog, is a system data base containing information about base tables, views, access rights, user-ids, etc. that can be queried through the use of SQL SELECT statements (Hursch & Hursch, 1988).

Functions of SQL

The relational sublanguage, SQL, provides support for three general functions. First, SQL acts as a Data Definition Language (DDL), which is used for defining the structure of the data. Second, it serves as a Data Manipulation Language (DML), which is used for modifying data within the data base. Finally, SQL is a Data Control Language (DCL), which controls user access by specifying security constraints. By providing these three general functions, SQL allows sophisticated data management processes to be performed on data bases that are based upon highly orthogonal yet simple principles.

SQL Relational Operators

The tools used for performing these data management processes are termed relational operators. The operators that are supported by the relational model are UNION, INTERSECTION, DIFFERENCE, PRODUCT, PROJECTION, JOIN, and SELECT. Of these operators, only PRODUCT, PROJECTION, JOIN, and SELECT may be performed on tables with differing structures.

SQL Keys and Keywords

Two additional instruments provide for searching and query construction in the SQL environment: keys and key words. Since the rows in a relational data base are unordered, and efficient searches are of supreme importance, a device must exist for rapidly locating desired data. This device, called a key, performs this function. An individual column or columns may be designated as the key(s) for a particular table which requires that each value in the key column be unique. This ensures that searching the data base takes place in an expedient, rather than random fashion.

Finally, like most programming languages, SQL retains a list of several words that may not be used in tables or column names. Figure 1 lists SQL's key words (Trimble & Chappell, 1989).

Framework for SQL Extensions

The many powerful characteristics of SQL are the reason that it is an industry standard and is the primary motivation for using this language as the basis for the construction industry-specific extensions. In creating these extensions the purpose of this research is not to develop a new relational query language, to develop a new software package, or to completely revamp the most widely accepted relational data base query language. The intent is to develop a new conceptual model of construction industry-specific extensions upon which subsequent research and applications can be generated. This research focuses on industry-specific extensions that have not yet been addressed. The primary benefit for the construction industry will be improved productivity in the area of information retrieval. The productivity gains will be realized through the creation of simple, more effective methods of querying relational data bases. By adding construction-specific vocabulary, construction industry could begin to tap into the power and efficiency of the standard in relational query languages.

ALL	AND	ANY	AS
ASC	AUTHORIZATION	AVG	BEGIN
BETWEEN	BY	CHAR	CHARACTER
CHECK	CLOSE	COBOL	COMMIT
CONTINUE	COUNT	CREATE	CURRENT
CURSOR	DEC	DECIMAL	DECLARE
DELETE	DESC	DISTINCT	DOUBLE
END	ESCAPE	EXEC	EXISTS
FETCH	FLOAT	FOR	FORTRAN
FOUND	FROM	GO	GOTO
GRANT	GROUP	HAVING	IN
INDICATOR	INSERT	INT	INTEGER
INTO	IS	LANGUAGE	LIKE
MAX	MIN	MODULE	NOT
NULL	NUMERIC	OF	ON
OPEN	OPTION	OR	ORDER
PASCAL	PLI	PRECISION	PRIVILEGES
PROCEDURE	PUBLIC	REAL	ROLLBACK
SCHEMA	SECTION	SELECT	SET
SMALLINT	SOME	SQL	SQLCODE
SQLERROR	SUM	TABLE	TO
UNION	UNIQUE	UPDATE	USER

Figure 1 SQL Keywords

Previous SQL Extensions

Several previous efforts have been directed at extending SQL, particularly in the representation of spatial data. In 1985 Sikeler proposed that SQL be extended to encompass the treatment of spatial relations and the use of a picture list to manage graphical output. That same year Roussopoulos (Egenhofer, 1989) developed PSQL (Pictorial SQL) in which two clauses were added to the SELECT-FROM-WHERE construct. In his work, Egenhofer criticizes this extension for making "the formulation of queries unnecessarily complicated". In 1987 Ingram added syntax extensions to SQL to address the needs of geographic information systems. The following year Herring (Egenhofer, 1989) pursued an object-oriented approach to extending SQL but sacrificed some SQL principles in the process. In 1989 Egenhofer attempted unsuccessfully to combine spatial concepts with a graphical user interface. By his own admission his work culminated in "a negative demonstration of the extension of an SQL-like language for spatial data handling". Although several attempts have been made at extending SQL in application specific areas, no published undertakings exist in handling construction information.

Nature of CI-SQL

Unlike some of the works cited in the previous section, CI-SQL will preserve all of the functionality and regulations of the ANSI supported version. Complete compatibility is maintained with existing implementations that use the ANSI standard. This is essential to ensure that this work is a true extension which augments rather than replaces an existing standard. The additions proposed here will be in the form of semantic rather than syntactical changes. This constraint means that instead of altering the structure of SQL commands and keywords, construction-specific vocabulary will be used to create new commands that adhere to the existing SQL pattern. Because this extension occurs in an area in which SQL rarely has been applied, the existing field of compatible relational data bases will be meager. For this reason a generic data base structure will be explained using a construction material example for illustration.

The role that CI-SQL plays in providing information retrieval facilities for the construction industry is that of an intermediary. CI-SQL begins the process of bridging the gap between construction data and the construction professional. In order for a user to access a data base, a query language must be provided as the tool for achieving this access. CI-SQL acts as the intermediary between the user and the data, possibly through the use of a user interface. The user interface would provide a more intuitive method of accessing the data without requiring the user to become an expert at formulating SQL queries. This interface would also address the issue of user-friendliness.

User-Defined Functions

In addition to providing embedded construction-specific functions, one of the most desirable extensions to SQL is the capability of creating construction-specific, user-defined data types, objects, and functions. Due to the specialized nature of construction data bases and queries, the ability to construct and reuse functions that are application specific is of enormous benefit to the end user. By providing this facility the queries can be tailored to construction industry data bases as well as the individual query needs of the user.

During the design phase of a construction project, project specifications are created to detail the materials and methods to be used during construction to provide the level of quality desired by the owner and his or her architect. In the process of creating these specifications the designer may enumerate literally thousands of products or materials and the particular characteristics that they should possess. The nature of this process lends itself very well to the use of object data typing since they have attributes just as the physical objects they represent. Furnishing the data base user with object data typing facilities and user-definable functions will allow for the construction of complex industry-specific queries that are not currently provided for in SQL.

In creating a user-defined function the following steps are required:

(a) Define the object type

(b) Create an instance or instances of the object type.

The initial step in creating a user-defined function that maintains the properties of the object oriented data base management system is the object type definition. This step creates the template or class from which other objects having the same properties can be created. In this example several of the critical attributes of wood doors are used to create the object type WoodDoor. It should be noted that in these constructions, existing SQL vocabulary is shown in uppercase letters, while additions to the syntax are shown in bold type. Object types begin with a capital letter whereas their instances begin with a lowercase letter. After each attribute or field is defined, its data type is characterized. For instance, in the following example the attribute Manufacturer is defined as a character string of up to fifteen characters (CHAR(15)).

```
CREATE type WoodDoor
    (Manufacturer CHAR(15),
    Model# CHAR(10),
    Face_material CHAR(20),
    Core_material CHAR(20),
    Edge_material CHAR(20));
```

Once the object type has been created the user is able to create actual objects, or instances, of the object type. Here, an instance of WoodDoor is woodDoor and is created as follows:

```
CREATE woodDoor
    instance ('Pella', 'SCO-30', 'oak', 'particle board', 'multi-ply
    laminated')
```

This process of creating an instance of WoodDoor, as with any object type, is accomplished much like the insertion of a row into a conventional relational table. However, instead of using the keyword INSERT, CREATE is used since the concept of "creating" an instance, or object, may be much more intuitive to the user. For each of the attributes defined in the creation of the object type, or class, the instance command inserts the subsequent values into the type definition parameters.

The final step in providing a user-defined function is the procedure of creating the function itself. The justification for such a development and addition to the existing body of SQL functions is that the ability to create application specific functions would greatly enhance the power and usability of SQL for industry professionals. Since the purpose of SQL is information retrieval, the more a user can customize the query language to meet his informational needs, the more powerful that language becomes. Thus, "the goal is to make the modeling of information as direct and natural as possible, and to overcome the impedance mismatch with programming languages that already have many of these richer facilities" (Beech, 1989). For this reason a suggested procedure for creating user-defined functions is given below. Although the example presented here may be limited in scope it demonstrates the potential application of user-defined functions in directly meeting an industry-specific need for information retrieval. If, as is common practice, an architect intended to specify the doors in a project based upon the materials that composed the face, core, and edge, or stile, of the door, the following function would allow him to search a construction data base and retrieve all products that met his criteria.

```
CREATE function select_door (face, core, edge) as
    SELECT manufacturer, model#
    FROM Division_8_Doors&Windows
    WHERE face_material = face
    AND core_material = core
    AND edge_material = edge
```

By specifying the arguments face, core and edge, this function would return the manufacturer and model number of all products in the

Division_8_Doors&Windows data base that met the given criteria.

By developing a robust collection of such user-defined, industry-specific functions, the procedure of retrieving data in a vertical market data base could be thoroughly customized. This customization could thus enhance and simplify an industry-specific query language to the point that knowledge of the industry's vernacular could largely replace a detailed knowledge of SQL syntax. The net result of this simplification would be that the information contained in the data base would be brought much closer to the end user.

SQL3 Function Definition

In its current state of development, SQL3 function definitions are accomplished as follows:

```
<SQL function> ::=
    [<function type>] FUNCTION <SQL function name>
        <parameter declaration list>
    RETURNS <SQL function result>
    <SQL statement>;
    END FUNCTION
```

where

```
<function type> ::= CONSTRUCTOR | ACTOR | DESTRUCTOR
<parameter declaration list> ::= <parameter declaration>
                              | (<parameter declaration> [,...])
<parameter declaration> ::= <parameter name> <data type>
                         | SQLCODE
                         | SQLSTATE
```

The following restrictions are placed upon these functions:
(a) all SQL functions with the same name must have the same corresponding parameter modes
(b) all SQL functions must contain a RETURN statement
(c) <function type> must be used in Abstract Data Type (ADT) definition and nowhere else
(d) constructor functions must have appropriate <new statement>
(e) destructor functions must have appropriate <destroy statement>

Syntax Conventions

Additionally, the following syntax conventions apply to all extensions contained in the following sections:
(a) function names are italicized and not capitalized

(b) object types or classes, as well as data base names, are capitalized
(c) data base fields or object instances are not capitalized

Construction Specific SQL Extensions

The first major domain of construction-specific SQL extensions has been developed to address the unique query needs of construction material and product selection. Due to the complex nature of the data base itself, the need for vernacular query capabilities is significant. Merely organizing the data does not sufficiently meet the informational needs of the construction professional. The ability to query the data base using vocabulary currently familiar to the typical user is necessary in order to maximize the use of this powerful relational tool. Using terminology very common to actual project specifications, these extensions were developed to provide construction-specific functions for information retrieval.

CI-SQL Extensions

(a) complies_with_standard

The first SQL extension created for construction industry-specific queries is the *complies_with_standard* function which would retrieve all standards to which a particular product conforms. The practical application of this function is twofold. During the design phase, the architect/engineer would be able to use this function to determine which products in a manufacturer's data base meet a particular design standard that he or she has specified. Additionally, during the bidding phase of the project, the contractor could use the same query function on a different data base containing pricing and availability information to determine the cost and availability of a particular product that has been specified by the design team. By providing the ability to return both the standards to which a particular material complies as well as the materials that meet a particular standard, the search capabilities for this scenario would be thoroughly covered. The following syntax extension shows how the first of two extensions are defined by using the particular material in question as the parameter or argument for the function and returning the standard or standards to which it conforms.

Syntax:

complies_with_standard ::=

ACTOR FUNCTION *complies_with_standard* (material REF (Materials))
RETURNS standard;

RETURN SELECT manufacturer, product, conformance

```
        FROM Materials
        WHERE product = material
    END FUNCTION
```

Sample Query:
Given a data base Division_9_Finishes with an instance acousticalCeilingTile, list the standard with which Armstrong's ceiling tile, model ACT-22M, complies.

Query Form:

```
SELECT *
FROM Division_9_Finishes
WHERE complies_with_standard (ACT-22M)
```

Query Result:

Manufacturer	Product	Conformance
Armstrong	ACT-22M	ASTM E 1264

(b) material_complies_with

The second SQL extension, *material_complies_with* is actually a variation of the *complies_with_standard* function. In this instance the function uses the specified standard as the argument and returns a list of materials meeting that standard. In the construction industry the application of this function would be in the retrieval of all appropriate materials conforming to a standard specified in the project specifications. For example, if a contractor bidding on a job desired to see a list of all fan motors that conform to a particular UL standard, the following function would provide the necessary capabilities for retrieving the acceptable options. The contractor could then choose from those options while considering such issues as cost and projected delivery time or add such restrictions as predicates in the WHERE clause of the query.

Syntax:

material_complies_with ::=

```
ACTOR FUNCTION material_complies_with (standard REF (Standards))
RETURNS material;

RETURN SELECT conformance, manufacturer, product
        FROM Materials
        WHERE conformance = standard
    END FUNCTION
```

(c) tested_per_test

The third construction-specific SQL extension, *tested_per_test*, provides the capability of determining whether a specified product has been subjected to a given test. For example, when specifying fire resistant wire glass, the architect/engineer might require that all wire glass products be tested against UL 9 in order to ensure proper performance in the case of fire. Similarly, the project owner might have a desire to incorporate a particular manufacturer's product with which he has had a high degree of success on previous projects. This function would allow the project owner to determine whether his chosen products met the requirements of the product testing specified by the designer.

Syntax:

tested_per_test ::=

```
ACTOR FUNCTION tested_per_test (manufacturer, material REF (Materials))
RETURNS testing_test;

    RETURN SELECT manufacturer, product, test
        FROM Materials
        WHERE product = material
    END FUNCTION
```

(d) material_tested_per

The fourth extension, *material_tested_per*, is a variation of the *tested_per_test* syntax. However, this implementation returns the material or materials that have passed a particular test rather than returning the test against which the material has been tested. The application here is clear in that during the bidding phase of a construction project, a contractor could search a construction data base, retrieving all materials of a given type that have passed the tests specified by the designer. This process would allow the contractor to maximize his profit on the material costs of a job by incorporating materials of equal quality but lower cost than those specified by the architect/engineer.

Syntax:

material_tested_per ::=

```
ACTOR FUNCTION material_tested_per (test)
RETURNS material;

    RETURN SELECT test, manufacturer, product
            FROM Materials
            WHERE test = test
    END FUNCTION
```

(e) meets_or_exceeds

Another material constraint common to project specifications is that a certain attribute of a product must meet or exceed a given value. To address this need, the CI-SQL function *meets_or_exceeds* has been developed. When passed the parameters of product type, attribute, and value, this function returns all products whose specified attribute is greater than or equal to the specified value. As an example, the project architect commonly specifies that a building's heating and air conditioning system must have an EER (energy efficiency ratio) that meets or exceeds a given value.

Syntax:

```
meets_or_exceeds ::=
ACTOR FUNCTION meets_or_exceeds (attribute, value)
RETURNS material;

    RETURN SELECT manufacturer, product, attribute, value
            FROM Materials
            WHERE attribute ≥ value
    END FUNCTION
```

Sample Application

The purpose of this section is to demonstrate the use of several of these extensions in a combined form. This demonstration will be designed to mimic an actual query that might be required in industry practice. It should be noted that in a total solution a user interface would hide much of the query construction from the user. Again, the data base that would be used for this query is only hypothetical and an actual industry data base might differ significantly depending on the input obtained from product manufacturers and specifiers.

Sample Scenario: The chief estimator for XYZ Construction Company is bidding on a school expansion. He has determined from the project drawings that sixteen wood doors will have to be supplied in one wing of the school. These doors must have the following characteristics:

1) Solid Core
2) 1-3/4" Thick
3) 3'0" Wide
4) AWI Grade III
5) Cost less than $250.00 each
6) Primer finish
7) Comply with ASTM E 221
8) Tested per ASTM E 81-S
9) Fire rating that meets or exceeds 1-1/2 hours
10) Acceptable manufacturers are:
    a. Allied Wood Products
    b. TreeCo Corporation
    c. Wood Builders Products Corp.
    d. Florida Door

Based on this information the estimator constructs a query to search the WoodDoors data base for all doors matching the above specifications. The query is structured as shown below.

```
SELECT manufacturer, model, cost
FROM WoodDoor
WHERE  core = 'Solid' AND
    thickness = '1-3/4"' AND
    width = '3'-0"' AND
    awi_grade = 'III' AND
    cost < 250 AND
    finish = 'primer' AND
    complies_with_standard (ASTM E 221) AND
    material_tested_per (ASTM E 81-S) AND
    meets_or_exceeds (fire_rating, 1.5) AND
    manufacturer = 'Allied Wood Products' OR
                   'TreeCo Corporation' OR
                   'Wood Builders Products' OR
                   'Florida Door' AND
    quantity_on_hand ≥ 16
ORDER BY manufacturer, model;
```

The result of this query might be as follows:

Manufacturer	Model	Cost
Allied Wood Products	JL 211-2	228.50
Allied Wood Products	JL 212-5	244.65
Allied Wood Products	JL 297-5	249.85

TreeCo Corporation	TCWD 1156	219.38
Wood Builders Products	W-8972Z	249.99
Wood Builders Products	W-8978Z	235.75

The primary benefit to such query capabilities would be that if a "live" data base were maintained by local or even national suppliers, the architects and contractors would have computerized access to detailed product information. Other beneficial options could be added to the system such as the ability to provide cut sheets, installation information, and pricing structures that depend upon passwords.

CONCLUSIONS

The primary results of this research are twofold. First, several specific SQL extensions were created which afford the potential for the development of many other related functions. This expansion clearly could be cultivated into a robust collection of construction applications. Second, the development process itself entailed difficulties that yielded unanticipated results.

This research should be viewed as the first of multiple passes necessary to create a complete and full-featured construction-specific superset of SQL. One of the additional passes necessary to providing a complete version of CI-SQL would be the expansion of construction-specific vocabulary. The development of many other pertinent construction vocabulary words could be achieved through the creation of a consortium of design professionals, construction management personnel, and construction materials experts. The goals of such a consortium should be aimed at developing all vocabulary that would increase productivity by eliminating much of the duplicate effort mentioned earlier. By continuing the process initiated by this research, the tools available for meeting the informational needs of construction professionals can be greatly enhanced. Given the fact that other industries have proprietary vocabularies, the procedure used in this research could be replicated in almost any other industry.

Clearly, future construction use will depend on the future research and release of SQL products. For example, once date/time functions are supported by SQL, construction applications such as calculating expected material delivery dates will be feasible. Also, many scheduling operations such as calculating activity float times, and projected project completion dates will be possible. Other construction activities such as cost estimating and bid analysis should be examined for applicable uses and extensions of SQL.

In addition to building CI-SQL based on future research, extensive data base development would be required in order to provide the construction industry with critical data such as pricing and availability of products. The development of such data bases would depend largely upon the participation of manufacturing groups to furnish detailed product information. This information would have to include attribute templates, or models of all the data that uniquely identify individual products or groups of products. For example, the attribute template for doors might include such fields as size, swing, thickness, material, lites, style, and fire rating. Characteristics such as these would allow an architect or contractor to specify the values for each of these attributes that uniquely identify doors. Thus, participation on the part of manufacturers would be crucial to the success of building large construction materials data bases that could be used by the construction industry to increase productivity.

References

Beech, D(1989, February 15). The future of SQL. *Datamation*, pp 45-48.

Codd, EF(1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), pp 377-387.

Date, CJ(1989). *A guide to the SQL standard* (2nd ed.). Reading, MA: Addison-Wesley Publishing Company, pp 1.

Egenhofer, MJ(1989). *Spatial query languages*. Dissertation Abstracts International, 51, 5104A. (University Microfilms No. 9023850), pp 79-80.

Hursch, CJ & Hursch, JL(1988). *SQL: The structured query language*. Blue Ridge Summit, PA: Tab Books Inc., pp 9-10.

Pascal, F(1989, September). A brave new world? *Byte*, pp 247-256.

Trimble, JH & Chappell, D(1989). *A visual introduction to SQL*. New York: John Wiley & Sons, pp 3-12.

Wipper, F(1989). *Guide to DB2 and SQL/DS*. New York: McGraw-Hill Publishing Company, pp 27.5