Index Only Tables

Standard Oracle 7.X involves keeping indexes as a B^+-tree that contains pointers to data blocks (see Chapter 6). This gives good performance in most situations. However, both the index and the data block must be accessed to read the data. Moreover, key values are stored twice—in the table and in the index—increasing the storage costs. Oracle 8 supports both the standard indexing scheme and also **index only tables,** where the data records and index are kept together in a B-tree structure (see Chapter 6). This allows faster data retrieval and requires less storage space for small-to medium-sized files where the record size is not too large.

Partitioned Tables and Indexes

Large tables and indexes can be broken down into smaller partitions. The table now becomes a logical structure and the partitions become the actual physical structures that hold the data. This gives the following advantages:

- Continued data availability in the event of partial failures of some partitions.
- Scalable performance allowing substantial growth in data volumes.
- Overall performance improvement in query and transaction processing.

13.4 An Overview of SQL3

We introduced SQL as the standard language for RDBMSs in Chapter 8. As we discussed, SQL was first specified in the 1970s and underwent enhancements in 1989 and 1992. Chapter 8 covered the syntax and facilities of SQL-92, also known as SQL2. The language is continuing its evolution toward a new standard called SQL3, which adds object-oriented and other features. We already illustrated through various examples in Informix Universal Server and Oracle 8 how SQL can be extended to deal simultaneously with tables from the relational model and classes and objects from the object model. This section highlights some of the features of SQL3 with a particular emphasis on the object-relational concepts.

13.4.1 The SQL3 Standard and Its Components

We will briefly point out what each part of the SQL3 standard deals with, then describe some SQL3 features that are relevant to the object extensions to SQL. The SQL3 standard includes the following parts (Note 17):

- SQL/Framework, SQL/Foundation, SQL/Bindings, SQL/Object.
- New parts addressing temporal, transaction aspects of SQL.
- SQL/CLI (Call Level Interface).
- SQL/PSM (Persistent Stored Modules).

SQL/Foundation deals with new data types, new predicates, relational operations, cursors, rules and triggers, user-defined types, transaction capabilities, and stored routines. SQL/CLI (Call Level Interface) provides rules that allow execution of application code without providing source code and avoids the need for preprocessing. It provides a new type of language binding and is analogous to dynamic SQL in SQL-92. Based on Microsoft ODBC (Open Database Connectivity) and SQL Access Group's standard, it contains about 50 routines for tasks such as connection to the SQL server, allocating and deallocating resources, obtaining diagnostic and implementation information, and controlling termination of transactions. SQL/PSM (Persistent Stored Modules) specifies facilities for partitioning an application between a client and a server. The goal is to enhance performance by minimizing network traffic. SQL/Bindings includes Embedded SQL and Direct Invocation as in SQL-92. Embedded SQL has been enhanced to include additional exception declarations. SQL/Temporal deals with historical data, time series data, and other temporal extensions, and it is being proposed by the TSQL2 committee (Note 18). SQL/Transaction specification formalizes the XA interface for use by SQL implementors.

13.4.2 Some New Operations and Features in SQL3

New types of operations have been added to SQL3. These include SIMILAR, which allows the use of regular expressions to match character strings. Boolean values have been extended with UNKNOWN when a comparison yields neither true nor false because some values may be null. A major new operation is **linear recursion** for specifying recursive queries (see Section 7.6). To illustrate this, suppose we have a table called PART_TABLE(Part1, Part2), which contains a tuple <p1, p2> whenever part p1 contains part p2 as a component. A query to produce the **bill of materials** for some part p1 (that is, all component parts needed to produce p1) is written as a recursive query as follows:

```
WITH RECURSIVE

BILL_MATERIAL (Part1, Part2) AS

(SELECT Part1, Part2

FROM PART_TABLE

WHERE Part1 = ‘p1’

UNION ALL

SELECT PART_TABLE(Part1), PART_TABLE(Part2)

FROM BILL_MATERIAL, PART_TABLE

WHERE PART_TABLE.Part1 = BILL_MATERIAL(Part2))

SELECT * FROM BILL_MATERIAL

ORDER BY Part1, Part2;
```

The final result is contained in BILL_MATERIAL(`Part1`, `Part2`). The UNION ALL operation is evaluated by taking a union of all tuples generated by the inner block until no new tuples can be generated. Because SQL2 lacks recursion, it was left to the programmer to accomplish it by appropriate iteration. We discuss recursion in relational queries in more detail in Chapter 25.

For security in SQL3, the concept of **role** is introduced, which is similar to a "job description" and is subject to authorization of privileges. The actual persons (user accounts) that are assigned to a role may change, but the role authorization itself does not have to be changed. SQL3 also includes syntax for the specification and use of **triggers** (see Chapter 23) as active rules. Triggering events include the INSERT, DELETE, and UPDATE operations on a table. The trigger can be specified to be considered BEFORE or AFTER the triggering event. This feature is present in both of the ORDBMS systems we discussed. The concept of **trigger granularity** is included in SQL3, which allows the specification of both row-level triggers (the trigger is considered for each affected row) or statement-level trigger (the trigger is considered only once for each triggering event) (Note 19). For distributed (client-server) databases (see Chapter 24), the concept of a **client module** is included in SQL3. A client module may contain externally invoked procedures, cursors, and temporary tables, which can be specified using SQL3 syntax.

SQL3 also is being extended with programming language facilities. Routines written in computationally complete SQL with full matching of data types and an integrated environment are referred to as **SQL routines.** To make the language computationally complete, the following programming control structures are included in the SQL3 syntax: CALL/RETURN, BEGIN/END, FOR/END_FOR, IF/THEN/ELSE/END_IF, CASE/END_CASE, LOOP/END_LOOP, WHILE/END_WHILE, REPEAT/UNTIL/END_REPEAT, and LEAVE. Variables are declared using DECLARE, and assignments are specified using SET. **External routines** refer to programs written in a host language (ADA, C, COBOL, PASCAL, etc.), possibly containing embedded SQL and having possible type mismatches. The advantage of external routines is that there are existing libraries of such routines that are broadly used, which can cut down a lot of implementation effort for applications. On the other hand, SQL routines are more "pure," but they have not been in wide use. SQL routines can be used for server routines (schema-level routines or modules) or as client modules, and they may be procedures or functions that return values.

A number of built-in functions enhance the capability of SQL3. They are used to manage **handles,** which in turn are classified into **environment handles** that refer to capabilities, **connection handles** that are connections to servers, and **statement handles** that manage SQL statements and cursors. There are also functions to manage descriptors and diagnostics as well as help functions.

13.4.3 Object-Relational Support in SQL3

The SQL/Object specification extends SQL-92 to include object-oriented capabilities. New data types include Boolean, character, and binary large objects (LOBs), and large object locators. We saw in Section 13.3 how large objects are used in Oracle 8.

SQL3 proposes LOB manipulation within the DBMS without having to use external files (Note 20). Certain operators do not apply to LOB-valued attributes—for example, arithmetic comparisons, group by, and order by. On the other hand, retrieval of partial value, LIKE comparison, concatenation, substring, position, and length are operations that can be applied to LOBs.

Objects in SQL3

Under SQL/Foundation and SQL/Object Specification, SQL allows user-defined data types, type constructors, collection types, user-defined functions and procedures, support for large objects, and triggers. Objects in SQL3 are of two types:

- Row or tuple types whose instances are tuples in tables.
- Abstract Data Types (shortened as ADT or value ADT), which are any general types used as components of tuples.

A **row type** may be defined using the syntax

CREATE ROW TYPE row_type_name (<component declarations>);

An example is

CREATE ROW TYPE Emp_row_type (

name VARCHAR (35),

age INTEGER

);

CREATE ROW TYPE Comp_row_type (

compname VARCHAR (20),

location VARCHAR (20)

);

A table can then be created based on the row type declaration as follows:

CREATE TABLE Employee **OF TYPE** Emp_row_type;

CREATE TABLE Company **OF TYPE** Comp_row_type;

A component attribute of one tuple may be a **reference** (specified using the keyword REF) to a tuple of another (or possibly the same) relation. Thus we can define

```
CREATE ROW TYPE Employment_row_type (

employee REF (Emp_row_type),

company REF (Comp_row_type)

);

CREATE TABLE Employment OF TYPE Employment_row_type;
```

SQL3 uses a **double dot notation** to build path expressions that refer to the components of tuples. For example, the query below retrieves employees working in New York from the `Employment` table.

```
SELECT Employment..employee..name

FROM Employment

WHERE Employment..company..location = ‘New York’;
```

In SQL3, â is used for **dereferencing** and has the same meaning assigned to it in C. Thus if *r* is a reference to a tuple and *a* is a component attribute in that tuple, *r* â *a* is the value of attribute *a* in that tuple. **Object identifiers** can be explicitly declared and accessed. For example, the definition of `Emp_row_type` may be changed as follows:

```
CREATE ROW TYPE Emp_row_type (

name CHAR (35),

age INTEGER,

emp_id REF (Emp_row_type)

);
```

In the above example, the `emp_id` values may be system generated by using

CREATE TABLE Employee **OF TYPE** Emp_row_type

VALUES FOR emp_id **ARE SYSTEM GENERATED;**

If several relations of the same row type exist, SQL3 provides a mechanism by which a reference attribute may be made to point to a specific table of that type by using

SCOPE FOR <attribute> **IS** <relation>

Although the row types discussed above provide the functionality of objects and eventually allow construction of complex object types by combining row types, they do not provide for encapsulation as discussed in Section 11.3, which is an essential feature of object modeling. Encapsulation is provided through abstract data types in SQL3.

Abstract Data Types in SQL3

In SQL3 a construct similar to class definition is provided whereby the user can create a named user-defined type with its own behavioral specification and internal structure; it is known as an **Abstract Data Type** (**ADT**). The general form of an ADT specification is:

CREATE TYPE <type-name> (

list of component attributes with individual types

declaration of EQUAL and LESS THAN functions

declaration of other functions (methods)

);

SQL3 provides certain built-in functions for ADTs. For an ADT called `Type_T`, the **constructor function** `Type_T()` returns a new object of that type. In the new ADT object, every attribute is initialized to its default value. An **observer function** `A` is implicitly created for each attribute *A* to read its value. Hence, `A(X)` returns the value of attribute *A* of `Type_T` if *X* is of type `Type_T`. A **mutator function** for updating an attribute sets the value of the attribute to a new value. SQL3 allows these functions to be blocked from public use; an EXECUTE privilege is needed to have access to these functions.

An ADT has a number of user-defined functions associated with it. The syntax is

FUNCTION <name> (<argument_list>) **RETURNS** <type>;

Two types of functions can be defined: internal SQL3 and external. Internal functions are written in the extended (computationally complete) version of SQL. External functions are written in a host language, with only their signature (interface) appearing in the ADT definition. The form of an external function definition is

DECLARE EXTERNAL <function_name> <signature>

LANGUAGE <language_name>;

Many ORBDMSs have taken the approach of defining a set of ADTs and associated functions for specific application domains, and packaging them together. For example, the Data Blades in Informix Universal Server and the cartridges in Oracle can be considered as such packages or libraries of ADTs for specific application domains.

ADTs can be used as the types for attributes in SQL3 and the parameter types in a function or procedure, and as a source type in a distinct type. **Type Equivalence** is defined in SQL3 at two levels. Two types are **name equivalent** if and only if they have the same name. Two types are **structurally equivalent** if and only if they have the same number of components and the components are pairwise type equivalent. Under SQL-92, the definition of UNION-compatibility among two tables is based on the tables being structurally equivalent. Operations on columns, however, are based on name equivalence. Thus the operation

UPDATE table_t

SET c1 = c2

WHERE <condition>

would be acceptable if the types of c1 and c2 are name equivalent (or are implicitly convertible). Attributes and functions in ADTs are divided into three categories:

- PUBLIC (visible at the ADT interface).
- PRIVATE (not visible at the ADT interface).
- PROTECTED (visible only to subtypes).

It is also possible to define virtual attributes as part of ADTs, which are computed and updated using functions. SQL3 has rules for dealing with inheritance (specified via the UNDER keyword), overloading, and resolution of functions. They can be summarized as follows:

Inheritance

- All attributes are inherited.
- The order of supertypes in the UNDER clause determines the inheritance hierarchy.
- An instance of a subtype can be used in every context in which a supertype instance is used.

Overloading

- A subtype can redefine any function that is defined in its supertype, with the restriction that the signature be the same.

Resolution of Functions

- When a function is called, the best match is selected based on the types of all arguments.
- For dynamic linking, the runtime types of parameters is considered.

SQL3 supports constructors for collection types, which can be used for creating nested structures for complex objects. List, set, and multiset are supported as built-in type constructors. Arguments for these type constructors can be any other type, including row types, ADTs, and other collection types. Instances of these types can be treated as tables for query purposes. Collections can be **unnested** by correlating derived tables in SQL3 (Note 21). For example, to return the elements of the set hobbies for employee John Smith, we first define an attribute in the table `employee` as follows:

```
hobbies SET (VARCHAR(20))
```

We can then write

```
THE (SELECT e.hobbies
       FROM employee e
       WHERE e.name = "John Smith")
```

Another facility in SQL3 is the supertable/subtable facility, which is not equivalent to super and subtypes and no substitutability is assumed. However, a subtable inherits every column from its supertable; every row of a subtable corresponds to one and only one row in the supertable; every row in the supertable corresponds to at most one row in a subtable. INSERT, DELETE, and UPDATE operations are appropriately propagated. For example, consider the `real_estate_info` table defined as follows:

```
CREATE TABLE real_estate_info (

property real_estate,

owner CHAR(25),

price MONEY,

);
```

The following subtables can be defined:

```
CREATE TABLE american_real_estate UNDER real_estate_info;

CREATE TABLE georgia_real_estate UNDER american_real_estate;

CREATE TABLE atlanta_real_estate UNDER georgia_real_estate;
```

We have given an overview of the proposed facilities in SQL3. At this time, both the SQL/Foundations and SQL/Object specification have reached the third step of the standardization process called the Committee Draft status. It is evident that the facilities that make SQL3 object-oriented closely follow what has been implemented in commercial ORDBMSs. The next two steps of standardization are called Draft International Standard and International Standard, respectively. SQL/MM (multimedia) is being proposed as a separate standard for multimedia database management with multiple parts: framework, full text, spatial, general purpose facilities, and still image. It is being pursued by a separate committee. We already saw the use of the two-dimensional data types and the image and text Datablades in Informix Universal Server that have considered issues relevant to this standard.

13.5 Implementation and Related Issues for Extended Type Systems

Other Issues Concerning Object-Relational Systems

There are various implementation issues regarding the support of an extended type system with associated functions (operations). We briefly summarize them here (Note 22).