3 - Database Technologies and SQL -- A Short Tutorial[1]

3.1 Introduction

Databases are used in almost all ebusiness applications. These databases may use different database management systems from different vendors and may reside on different computers (microcomputers,

[1] Co-authored by Kamran Khalid

minicomputers, mainframes) which are interconnected through different networks. The focus of this tutorial is primarily on how different databases are used for different applications in contemporary enterprises. In this short and informal tutorial, we will give a broad perspective and highlight the database concepts and technologies as they relate to ebusiness. Due to the space limitations, it is not possible to include many details. Numerous books have been written on different aspects of databases by now. The following books (some are classics and have gone through several editions) are recommended for additional information:

- Date, C., "An Introduction to Database Systems", Addison Wesley
- Elmasri and Navathe, "Fundamentals of Database Systems", Benjamin Cummings
- Martin, D., "Advanced Database Techniques", MIT Press
- Teorey, T.J., "Database Design", Prentice Hall
- Hernandez, M. "Database Design for Mere Mortals : A Hands-On Guide to Relational Databases" Addison-Wesley, 1997
- Muller, J., "Database Design for Smarties: Using UML for Data Modeling", Morgan Kaufman, 1999

3.2 Database Management Concepts

3.2.1 Files and Databases

At the lowest level, a *data item* is the smallest unit of data which cannot be subdivided. Examples of data items are part_no, part_name, weight, cost, etc. A *data record* is a collection of data items. An example of a data record is a part record which is a collection of part_no, part_name, weight and cost of a part. A *file* is a collection of similar data records. For example, a customer file consists of customer records and a parts-file consists of part records. Examples of files, also called "flat files", are text files, html files, XML files, etc.

Conceptually, a *database* is a collection of files (dissimilar records). For example, a manufacturing database is a collection of data records and files associated with manufacturing activities (e.g., finished goods inventory, bill of materials, equipment), and a financial database consists of payroll data, accounts receivable, general ledger, etc. It is common to assign additional properties to a database definition. For example, according to Elmasri and Navathe [Elmasri 1989]:

"A database has the following implicit properties:

- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.
- A database represents some aspect of the real world. Changes to the miniworld are reflected in the database."

In a database environment, different users can view, access and manipulate the data in a database. The database may be a business database for financial and administrative applications; a manufacturing database containing bills of material, goods inventory and scheduling data; an office database consisting of memos and proposals; and a knowledgebase for artificial intelligence and expert system applications.

3.2.2 Database Management System (DBMS)

Database access and manipulation are controlled by a database management system (DBMS). A DBMS, shown in Figure 3-1, is a software package which is designed to

- Manage logical views of data so that different users can access and manipulate the data without having to know the physical representation of data
- Manage concurrent access to data by multiple users, enforcing logical isolation of transactions
- Enforce security to allow access to authorized users only
- Provide integrity controls and backup/recovery of a database.

These functions of a DBMS are described later. As shown in Figure 3-1, a typical database management system (DBMS) uses a database dictionary/directory to store the data views, data relationships, data formats and security restrictions; database logs to record the activities of transactions; and lock tables to allow synchronous concurrent access to the database by several users.

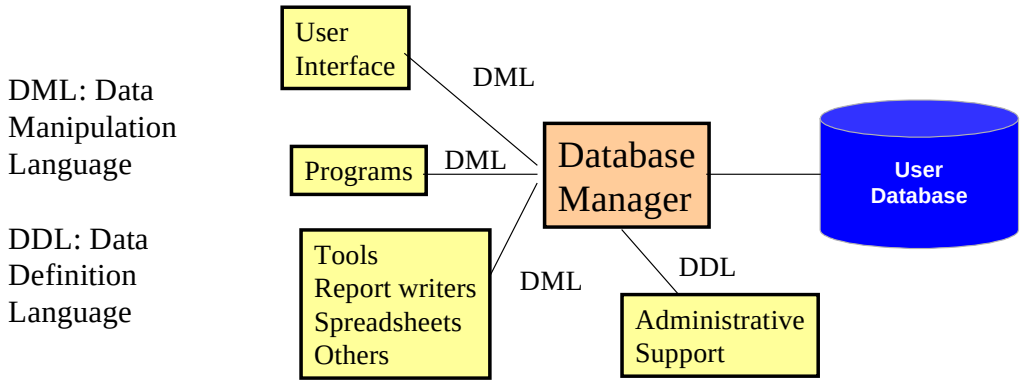


Figure 3-1: Database Conceptual View

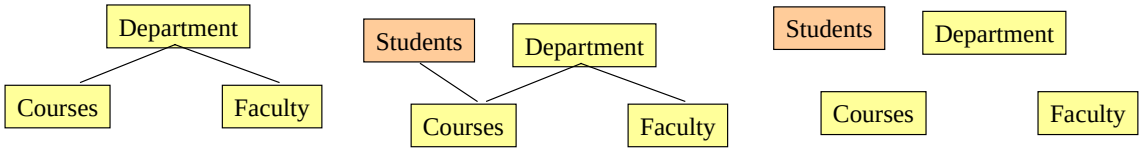
3.2.3 Data Models and Categories of DBMS.

A data model is a conceptual representation of data that does not include many of the details of how the data is physically stored. Database management systems have traditionally supported the following data models (see Figure 3-2):

- Hierarchical: the data model supports one to many relationships (i.e. each record has only one parent). The DBMSs which are based on this data model are called hierarchical DBMSs. IBM's IMS is an example of a hierarchical DBMS.
- Network: Many to many relationships among logical data records are supported. Example of a network database management system is Cullinets's IDMS (currently supported by Computer Associates).
- Relational: the data is viewed as tables (relations). Examples of relational DBMS are IBM's DB2, Oracle's RDBMS, and Microsoft's Access.

Two other data models gained importance since the early 1980s -- Entity-Relationship-Attribute (ERA) and Semantic data models. In the ERA data model, the data is viewed in terms of entities (objects), attributes of entities and relationships between the entities. This model is used to build a conceptual view of data in an organization (called logical data model). Semantic data model is an extension of the ERA model. The main difference is that the relationships carry meanings and the objects can inherit properties from other objects. "Object-oriented databases", discussed in section 3.6, use the semantic data model for storage and retrieval of objects and rules for complex engineering, business and expert systems applications. Other data models such as dimensional data models are used in specialized application areas such as data warehouses (see the chapter on data warehouses and data mining for a discussion of dimensional models)

Although database management systems can be categorized in a wide variety of ways, the categorization by data models is the most common. The network and hierarchical DBMS, not discussed in this tutorial, are older systems (these DBMSs flourished in the 70s). We will concentrate more on the relational and object-oriented DBMSs.



a: Hierarchical View b: Hierarchical View c: Relational View

Figure 3-2: Common Data Models

3.2.4 Data View Support

A DBMS allows different users to view the same data differently. For example, information about an employee can be viewed differently by different users. The term *schema* is used in database literature to represent a data view. In a DBMS, schema exist at three levels: internal, conceptual and external (see Figure 3-3). The *internal schema* shows the physical format (e.g., linked list) in which the data are stored on a storage medium. The *conceptual schema*: shows the logical layout and the relationships between data records of a database (it is also referred to as the logical data model). Database management systems have traditionally supported the conceptual schemas based on the hierarchical, network and relational data models. An *external schema*, also called a *.subschema*:, shows a user view of data. This view can be hierarchical, network, relational or object-oriented. In most cases, the external schema is a subset of the conceptual schema. However, relational views can be created from a hierarchical and/or network conceptual views.

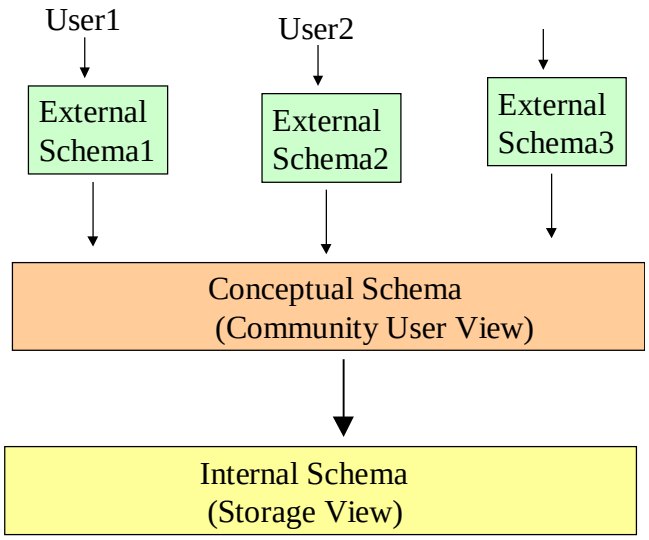


Figure 3-3: Schemas (Views) in a Database Environment

3.2.5 Data Definition Facilities

Data definition facilities allow creation of databases. A *Data Definition Language (DDL)* is used to define the data formats and data relationships. DDL allows data definitions at the conceptual as well as external schema levels. In addition, security and authority information is defined. DDLs can be interactive commands or batch programs. Results of DDL commands are stored in the database dictionary. In many corporations, a database administrator controls the data definition facilities for corporate-wide views, access rights, and enforcement of standards.

3.2.6 Data Manipulation Facilities

A.*Data Manipulation Language (DML)* allows a user to access, manipulate and modify the database. The power and capability of DMLs depends on the underlying DBMS. Most of the modern DMLs, such as SQL, support ad hoc queries which select information and display answers on demand. DML statements may be embedded in programs written programming languages such as C, C++, Java and Cobol. A DML may support report generators which produce reports with headings on special forms with appropriate printer controls. Some specialized packages such as spreadsheets, simulation packages and expert system shells may provide interfaces with database DMLs. For example, the Lotus Data Lens allows Lotus-123 spreadsheets to access relational databases through SQL. In addition, special features, such as graphics, can be built around a DML.

3.2.7 Operational Facilities

The operational facilities of a DBMS provide security, integrity and backup/recovery of a database. This includes authentication, audit trails, data consistency (the data must correctly reflect the state of a system even after failures), and concurrency (the data must be simultaneously accessible by different users). Operational facilities of a large, centralized DBMS must be comprehensive enough to allow simultaneous access of hundreds of users to large centralized databases. On the other hand, extensive operational facilities may not be needed for single user microcomputer databases.

Different commercial DBMSs provide different levels of data views, data definitions, data manipulation, and operational support. An area of active work is support of databases over a network where a database can be accessed by a wide range of programs and users across a network (see Figure 3-4). In such cases, ODBC/JDBC (Open Database Connectivity/Java Database Connectivity) software is typically used for access of databases from remote clients (programs or user interafces) ODBC/JDBC are discussed in the “Distributed Data and Transaction Management Chapter”.

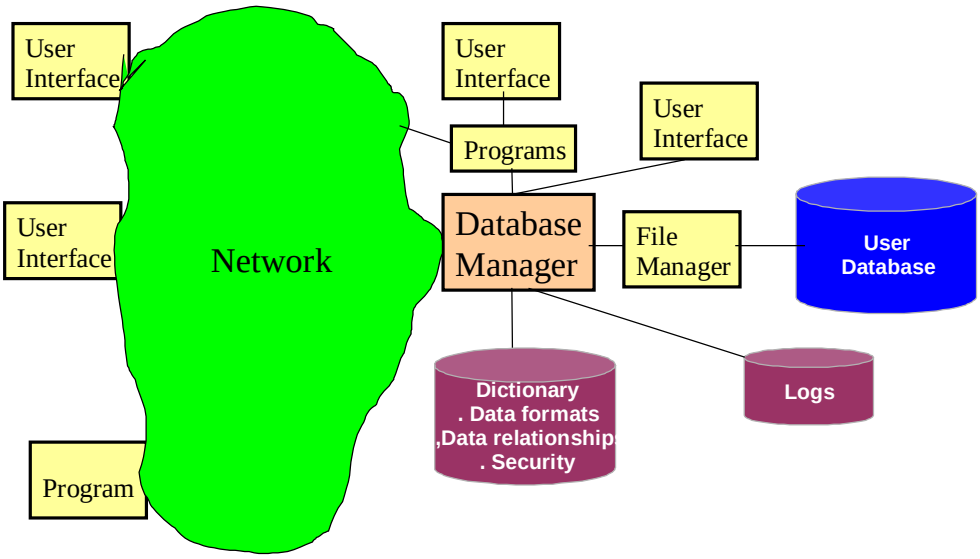


Figure 3-4: Database Management Over a Network

3.3 Overview of Relational Databases

The relational database technology, introduced by E.F. Codd [Codd 1970] at IBM, views all data as tables; all database operations are on tables; and all outputs produced are also tables. A relational database is a collection of tables. Figure 3-5 shows a relational database which consists of two tables: EMPLOYEE and OFFICE. The following terms are used in relational DBMSs (RDBMSs):

- A relation is a table in which each row is unique. In addition, a relation must have a fixed number of columns. A table which satisfies these two properties is known to be in the "First Normal Form". The EMPLOYEE and OFFICE tables represent two relations in First Normal Form.
- A tuple of a relation is synonmous to a row in a table. For example, the EMPLOYEE relation has 4 tuples.
- An attribute of a relation is a table column. For example, the OFFICE relation has 3 attributes: Location, Manager and Phone.
- The degree of a relation represents the number of attributes in a table. For example, the degree of OFFICE relation is 3.
- The domain of an attribute represents the range of values for an attribute. For example, the domain for the age attribute is 0 to, say, 120 years.

Name	Age	Salary (K)	Location
Joe	35	42	NY
Pat	29	60	LA
Bruce	25	42	Chicago
Sam	40	75	NY

Manager	Phone	Location
Donna	555-1000	NY
Roger	555-1111	LA
Dave	555-2222	Chicago

Figure 3-5: Relational Tables Employee and Office

The relational DBMSs allow a user to access information from the database with only three basic operations:

- Selection
- Projection
- Join

Selection chooses rows of a table based on a criteria. For example, selection on EMPLOYEE for Age > 30 produces the rows for Joe and Sam. Projection chooses columns of a table based on a criteria. For example, projection on EMPLOYEE for the Name column lists the names: Joe, Pat, Bruce, and Sam. Join combines two different tables on a common attribute. For example, join of the EMPLOYEE and OFFICE tables on LOCATION is shown in Figure 3-6 . Theoretically, a join between two relations r1 and r2 on the joining condition r1.a1 = r2.a2 involves the following steps (a1 and a2 represent two attributes):

- Form product of r1 and r2 to produce r3'. In a product (cartesian), every tuple of r1 is concatenated with every tuple of r2 so that r3' has m x n tuples if r1 has m tuples and r2 has n tuples.
- Perform a selection on r3' where the joining attributes a1 and a2 are the same. This produces r3'', known as :hp1.equijoin:ehp1..
- Eliminate duplicate attributes from r3'' with a projection. This produces r3, known as :hp1.natural join, :ehp1. or just a join. r3 is the normal result of a join.

Joins are implemented differently by different DBMSs for efficiency. In addition to the natural joins, other forms of joins are supported in relational DBMSs. Examples are the theta and outer joins. In theta joins, also known as non-equijoins, the joining condition is r1.a1 <> r2.a2. The outer joins retrieve rows that may not meet the join conditions. This allows retrieval of data that may be lost (e.g., if joining columns have null values).

Figure 3-6: Join of Employee and Office Tables

Name	Age	Location	Manager	Phone	Location
Joe	35	NY	Donna	555-1000	NY
Sam	40	75	Donna	555-1000	NY

In addition to the basic operations of selection, projection and join, relational DBMSs allow unions, differences and intersections. A union concatenates the tuples from r1 with r2 and produce r3. Duplicate relations are eliminated from r3 as a result of union. In addition, r1 and r2 must have same number of attributes and attributes must be from same domain (union compatible). After a difference between r1 and r2, the result r3 has tuples which occur in r1 and not in r2. After an intersection

between r1 and r2, the result r3 has tuples that are common in r1 and r2. Unions, intersection, differences, and products can be performed with selection, projection, and join.

In addition to these operations, some manipulation operations are introduced specifically for distributed systems. For example, the semi-join is introduced to minimize the internode traffic while performing a join of two remotely located tables [Bernstein 1981].

Relational DBMS's provide a number of attractive features:

- The relational model is simple and easy to understand.
- Desired data can be reached through a series of joins. If two tables do not have a joining column, then an "index table" can be created to facilitate joins.
- A standard query language, SQL, is used by all relational DBMS.
- Many commercial relational DBMSs are currently available for mainframes, minicomputers and workstations. Due to the popularity of relational DBMSs and SQL, many tools in business and engineering are developing interfaces to access the relational tables. This is leading to a corporate-wide database concept illustrated in Fig. E.2.
- The relational model supports "data independence" (i.e. the queries are non-procedural and do not have to know the physical data organization such as indexes and pointers).
- Relational database searches are based on data values and not on the position of data in the database. This makes data access easier.
- Relational databases and SQL are used in almost all of the currently available distributed database management systems (DDBMS).

However, relational DBMSs have a number of limitations:

- Relationships between tables cannot be modeled directly; each relationship is implicitly modeled by the inclusion of "foreign keys" as attributes. Simply stated, a foreign key enables a join between two relations. For example, the Employee-ID in the OFFICE table in Fig. E.6 is a foreign key.
- It is very difficult to represent complex design information in relational database model because relational tables do not lend themselves easily to complex data relationships, design versions and views.
- The user may be responsible for the semantic integrity of a query and completeness of an update. "Referential integrity", which ensures that all tables are modified correctly when a tuple is inserted or deleted, is not implemented in all relational DBMSs.
- The performance of queries depends on an "optimizer" which knows the internal structure of a database. It is difficult to know how well an optimizer is doing its job or if it is doing it at all.

3.4 SQL -- A Quick Overview

Structured Query Language (SQL) is the standard query language for relational databases. SQL, initially also referred to as "SEQUEL", was developed at the IBM San Jose Research Laboratories in 1974. It provides interactive ad hoc queries as well as program interfaces in C, C++, Java, Cobol, Fortran, ADA and PL1. The SQL language consists of a set of facilities for defining, manipulating, and controlling data in a relational database. Basically, SQL is at the core of relational da.tabase management systems (see Figure 3-7).

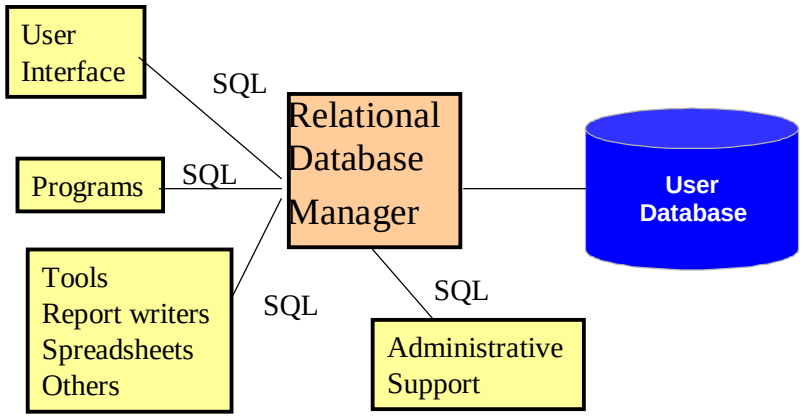


Figure 3-7: SQL is at the Core of Relational Databases

3.4.1 Data Definition

SQL data definition language (DDL) is used to create tables by using a CREATE TABLE command. The following two SQL statements are used to create a parts and a customers table:

CREATE TABLE parts (part_no char (4), part_name char(5)), price numeric(5))

CREATE TABLE customers (cust_name char(30) not null, address char(30) not null, cust_id char(12) unique not null, part_no char(4))

3.4.2 Data Retrieval

The main power of SQL lies in its data manipulation facilities. There are four basic SQL operations: SELECT, UPDATE, INSERT, and DELETE. All data retrievals are invoked by a SELECT command, which has the following general syntax:

SELECT <a1,a2,a3,...,an> FROM <t1,t2,...,tm> WHERE <conditions>;

where a1, a2,,, an are the attributes; t1, t2,, tm, are the tables; and the conditions, if specified, indicate the retrieval criteria. Conditions are specified by the "attribute op value" pairs, which can be combined through logical operators such as AND, OR, NOT. An op indicates predicates such as =, <, >, <=, >=, and <>. Examples of conditions are "age > 30", "age < 30 AND salary > 50K", etc. An SQL statement is terminated either by a ";" or by another SQL command. The SQL statements can be coded in upper or lower case. We will use uppercase letters to indicate the keywords in SQL.

The selection, projection and join operations of relational databases are performed by the SELECT statement. For example, the following statement performs relational selection (i.e., shows all columns):

SELECT * FROM t1 WHERE attribute op value

For example, " SELECT * FROM parts WHERE price > 100; " would display the rows of the parts table for prices more than 100. The statement, " SELECT * FROM parts; " would display the entire table. The projection and selection can be combined by using the following statement:

SELECT a1, a2,...,an FROM t1 WHERE attribute op value

For example, " SELECT part_no, part_name FROM parts WHERE price > 100 " would display the part_no and part_name from the parts table for prices more than 100. The joins are also performed by the select statement. The following statement causes an equijoin, where the joining condition is equality and the result include duplicates:

SELECT t1.*, t2.* FROM t1, t2 WHERE t1.a1 = t2.a2 op value;

The following statement invokes a natural join, an equijoin, which eliminates duplicates:

SELECT a1, a2, a3,...,an FROM t1, t2 WHERE t1.a1 = t2.a2 op value;

For example, " SELECT part_no, part-price, cust_name FROM parts, customers WHERE customer.part_no = parts_no " lists the customer names who have ordered certain parts. Additional conditions can be included in joins. For example, " SELECT part-price, cust_name FROM parts, customers WHERE part_no.customer = part_no.parts and part-price >200 " would list the names of the customers who have ordered parts which cost more than $200. A product between two tables is formed by ignoring the joining condition. For example, the following statement forms a product between the parts and customers tables: " SELECT * FROM parts, customers ". Theta joins are performed by the following statement:

SELECT a1, a2,...,an FROM t1, t2 WHERE a1.t1 <> a2.t1 op value

More than two tables can be joined in a single statement:

SELECT a1, a2,...,an FROM t1, t2,...,tm WHERE condition1 AND condition2 AND condition3;

A table can be joined with itself. Aliases can be used to avoid confusion. For example, the following statement produces a list of salesmen in the same city: " SELECT first.name, second.name FROM salesperson first = salesperson second WHERE first.city = second.city AND first.ss# <> second.ss# ". In this statement, first and second are assigned as aliases. You can build complex queries by nesting queries within other queries by using the following format:

SELECT a1, a2,...,an FROM t1 WHERE an IN (SELECT a5,a6,...,am FROM t2 WHERE am op value)

For example, the statement "SELECT part_no, part_name FROM parts WHERE part_no in (SELECT cust_name FROM customers WHERE city='Detroit')" would display part numbers and names ordered by the customers who live in Detroit. The innermost query is executed first; the outer query operates on the results of the inner query.

SQL provides a powerful set of built-in functions such as ORDER, AVG, SUM, COUNT, and GROUP BY. The statement " SELECT part_no, part_name FROM parts ORDER BY part_no; " lists the part_no and part_name, sorted by part_no. The statement " SELECT AVG (price), MIN(price), MAX(price), SUM (price), COUNT DISTINCT, COUNT (*) FROM parts; " lists the average, minimum, maximum, and sum of prices. This statement will also list the distinct and total count of records in the parts table. The statement " SELECT AVG(salary) FROM employees GROUP BY title; " will produce the following display:

Title Avg(salary) secretary 1300 programmer 2500 manager 3400

The MINUS produces a difference (this operator is supported by some DBMSs). For example, " SELECT * FROM parts MINUS (SELECT * FROM parts WHERE price <1000) " produces the list of parts with price greater than or equal to 1000.

The predicates, used in the where clause of the select statement, provide many options, such as the following:

Comparison: =, <, >, <=, >= , <>

BETWEEN/NOT BETWEEN: An example is "where price between 5 and 20;" * IN/NOT IN: example is "where price not in (5, 7, 10);"

LIKE/NOT LIKE: these are used for pattern recognition. A "_" is used for single characters, and "%" is used for 0 to n char length. For example,"select cust_name from customers where cust_name like 'B%';" displays the names of all customers whose name starts with a B. Like/not like predicates can be used with character or graphic data.

NULL: An example is "where part_no is null;"

3.4.3 Data Modification

SQL data modification statements allow insertion, deletion and update of data in tables through the INSERT, DELETE and UPDATE statements. Here are some (hopefully) self explanatory examples:

```
INSERT INTO parts(part_no, part_name, price) (xy22, rods, 100);

INSERT INTO parts(part_no, part_name, price) (xy22, rods, null);

INSERT INTO parts-high(part_no, part_name, price) (select part_no, part_name, price FROM
parts WHERE price >1000);

DELETE FROM parts where part_no = xy20;

UPDATE PARTS set price=120 where part_no=xy22;
```

3.4.4 View Support

Views may be used to operate on portions of tables. For example, " CREATE VIEW salesperson AS SELECT name, number FROM employees WHERE job='salesperson'; " creates a portion of the employees table populated with salesmen. The " DROP VIEW salesperson; " deletes the view. One view can be created to contain columns from several tables. Views are treated as tables in SQL and can be used in any of the SQL statements. For example, views can be created from joins and can be joined with tables or with other views. Views are created temporarily; the operations are performed on actual tables. Thus updates are performed on the tables from which the views are created. Views can be used to restrict user access and handle subqueries for intermediate tables. You may create a table for more permanent operations by using statements such as: " CREATE TABLE temp1 (name, part#, price) AS (SELECT name, part#, price FROM suppliers WHERE price > = 1000); ".

3.4.5 Administrative Facilities

SQL provides two data control statements for administrators:

```
GRANT access-type ON tablename TO id;

REVOKE access-type FROM id;
```

where access-type specifies: all privileges, update, select and insert. In addition, programmers can issue "commit work" command to make changes available to others. Before the commit command, only the person entering changes sees the changes. The "rollback work" command can be used to undo changes before commit. You can modify table structure (add columns, change column width) by using the following statements (these statements are not supported by ANSI SQL):

```
ALTER TABLE tablename ADD column-name datatype;

ALTER TABLE tablename MODIFY column-name datatype new-width;
```

3.4.6 Embedded SQL

SQL statements can be embedded in host programs written in several languages such as C, C++, Java, Cobol, Fortran, and PL1. The SQL statements in programs are embedded by using the EXEC SQL statements in a program:

```
EXEC SQL sql statements
```

Two unique problems are concerned with embedded SQL: connecting the SQL variables with programming language (host) variables and handling of multiple rows returned from SQL statements.

Connection with host variables is established by reading the selected attributes INTO a set of host program variables:

EXEC SQL SELECT a1, a2,,,, an INTO :p1, :p2,,,,,:pn FROM t1 WHERE condition;

The host variables p1, p2,,, pn are indicated by a ":". For example, the statement " EXEC SQL SELECT part_no part_name INTO :pnumber, :pname FROM parts " will store the part_number and part_name attributes from table parts into host variables pnumber and pname.

A "cursor" is used to handle multiple rows returned from SQL. The problem is that the traditional procedural languages are record oriented; they process one record at a time. However, SQL may return many rows as the result of a single embedded select statement. A cursor is first declared for an SQL statement to be executed. It is then opened in a manner similar to a file open. The program then issues FETCH statements to retrieve the rows returned. SQLCODE, a flag, is checked to see if any rows are left to be retrieved. The following code is an example:

EXEC SQL DECLARE cs CURSOR FOR <SQL statements>

EXEC SQL OPEN cs

EXEC SQL FETCH cs INTO <host variables>

CHECK SQLCODE for end of fetch

Figure 3-8 shows the sample pseudo C/C++ code which illustrates how embedded SQL can be used to extract information from the parts table defined by the shown CREATE command. The sample code has three segments. The first segment shows how the host variables are defined by using an int statement. The inclusion of SQLCA brings many of the SQL flags and variables (e.g., the SQLCODE which shows return codes) into the program code. The second segment of the code shows how the information for part number 75567 is accessed and printed. The third part of the code shows how multiple rows are processed by using the cursor statement. The reader should understand that the code shown here is given for illustrative purposes only. Differences exist between different languages and vendors.

Table definition:

```
CREATE TABLE parts (part_no numeric (4), part_name char(5)), part-price numeric(5)) ;
```

Sample code to Extract and display information from parts table;

```
Include SQLCA
Int p-number, p-price;
…
Exec SQL select part-no, part-price into :p-number, p-price from parts where part-no =75567;
Print (p-number, p-price);
…..
Exec SQL cursor CS
        select part-no, part-price into   where part-price >200;
Exec SQL select part-no, part-price into :p-number, p-price from parts where part-price =200;
Exec SQL open CS;
Do {Exec SQL Fetch CS into :p-number, p-price; } while sqlcode =0;
        Exec SQL close CS;
```

Figure 3-8: Sample Embedded SQL Code

3.4.7 Performance

SQL query optimization is the responsibility of DBMS which parses the query and then executes it in an appropriate manner. Clever techniques which rely on internal organization are not recommended. A user can create an index for fast access. For example, if the parts table needs to be accessed by part_no frequently, then an index on this column will speed up the performance. The following statements create and drop an index:

CREATE INDEX indexname ON tablename (column-name); DROP INDEX indexname ON tablename;

Many indices can be created on one table. The user does not specify when and how the index will be used. SQL determines when to use an index (recall that a SELECT statement does not include any reference to an index). An index is not used if a WHERE clause is absent in queries. The use of the index to satisfy a query is decided by the query optimizer.

Joins are the major source of performance problem. For an m row table join with an n table join:, there can potentially be mxn operations. Over the years, local optimizations for joins work fine, however, distributed joins can create major performance problems. In addition, you should minimize duplicate values to avoid too many updates.

3.4.8 SQL Products

At present, SQL is supported on a very large number of relational as well as non-relational database products. Here are examples of the major DBMSs which support SQL:

- SQL/DS, part of DB2, IBM's major relational DBMS
- Oracle from Oracle, Inc.
- Informix SQL from Informix Corp.
- Sybase from Sybase Corp. DbaseIV from Ashton Tate.
- SQL Access from Microsoft

In addition to the relational DBMS, SQL interfaces have been developed to read information from non-relational databases. Example is the EDA/SQL product from Information Builders, Inc., which uses SQL to retrieve information from more than 30 data sources such as ADABAS, IMS, VSAM, IDMS/R, Model 204, Supra, TOTAL, and OS/400 DB. SQL is also generated from a variety of tools to provide access to relational databases. Here are some examples:

- Data Lense which provides access to SQL databases from spreadsheets (e.g., the Lotus Data Lens from Lotus Corp).
- Executive information systems which access data through SQL (e.g., the Data Access Language for Command Center from Pilot Software).
- Expert systems which use SQL to access relational databases (e.g., the SQL interface for ART/IM System from Inference Corp.).
- CASE (computer aided software engineering) tools from Oracle, which are built around relational databases.

3.4.9 Strengths and Weaknesses

SQL has many strengths:

- SQL is easy to learn and use. It is based on relational algebra and allows a user to perform all retrievals by using a single verb (SELECT).
- SQL is the standard query language for all relational databases. An ANSI SQL standard has been published.
- At present, almost all relational database vendors support ANSI SQL. Due to its popularity, an application using ANSI SQL is portable across different database vendors running on different computer systems under different operating systems.
- SQL is used as the global query language for distributed heterogeneous databases. A user issues an SQL call which is translated to other data manipulation language, if needed.
- Due to the popularity of SQL, many tools and aids are being developed around SQL.

A practical limitation of SQL is that SQL queries can become quite complex when information from many tables is needed requiring many joins and nested queries. In addition, ad hoc SQL queries from inexperienced users can cause serious system performance problems. For example, a user may inadvertantly issue a join between 4 to 5 tables, causing thousands of messages in the system. E.F. Codd has discussed many weaknesses of SQL in his papers entitled "Fatal Flaws in SQL" <Codd 1988>. The main flaws discussed by Codd are as follows:

- SQL allows duplicate rows in tables,
- SQL supports an inadequately defined nesting of queries within queries, and
- SQL's support of third and fourth value logic (the logic that evaluates three and four conditions) is not adequate.

Suggested Readings. We have highlighted the main features of SQL. For a quick tutorial on SQL, refer to [Dowgiallo 1988]. SQL is an extensive query language with many features that are beyond the scope of this book. The following books are recommended for additional details:

- Date, D.J., "A Guide to The SQL Standard", Addison-Wesley, 1987
- Hursch, C. and Hursch, J., "SQL - The Structured Query Language", Tab Books, 1988
- Date, C.J., "A Guide to DB2", Addison Wesley, 1984
- Elmasri et al, “Fundamentals of database systems”, Addison-Wesley, 2001

3.5 SQL -- A Closer Look

In this section, we will look into SQL in more depth. There are certain differences in SQL as used by different DBMS vendors e.g., Oracle, DB2 by IBM or SQL Server by Microsoft etc. Here we will mostly discuss those SQL features that are used by almost all the platforms. Some exceptions are data types and some part of the ‘data administration’ portion. In these cases, we have followed Oracle notation.

SQL can be used both for data definition as well as data manipulation.

3.5.1 Data Definition

3.5.1.1 Create Relations (tables)

Tables or relations are created by CREATE TABLE command i.e. generically the tables are created as

CREATE TABLE table_name

(col_1 data_type (data_size) constraint1,…

col_2 data_type (data_size) constraint1,…

……)

e.g.,

Create table customers

```
CREATE TABLE customers
(cust_ID        number(6)       not null,
first_name      varchar2(20)    not null,
last_name       varchar2(20)    not null,
city            varchar2(20),
state           char(2),
zip             char(5),
date_register   date);
```

Create another table orders

```
CREATE TABLE orders
(orderID        number(5)       not null,
cust_ID         number (6),
```

Order_date date,

Order_val number (6));

the above statements will create Customers and orders tables having attributes(columns) mentioned in the statements. For each attribute, data type and size is mentioned. Data types may be different for different DBMS. For customers table, Cust_ID, first_name and last_name are declared not null. It means that these attributes can't have null values. We can also declare primary key, foreign keys and other constraints in create table command as we did for orders table. These issues will be discussed in some detail later in this section.

3.5.1.2 Drop Table

A table can be removed using command

DROP TABLE table_name;

Example:

DROP TABLE customers;

This command will not only delete all the tuples but will also remove the table structure. This command cannot be rolled back or undone.

The commands DELETE FROM TABLE and TRUNCATE TABLE are discussed under data manipulation.

3.5.1.3 Alter table

Table can be altered anytime after its creation. The change may be

- Addition of new columns

ALTER TABLE customers ADD (street char(15));

This statement will add a new column 'street' in the table customers

- Removing a Column

ALTER TABLE customers DROP COLUMN street;

This statement will remove street column from the customers table.

- Changing the data sizes etc.

ALTER TABLE customers MODIFY (last_name varchar(25));

This statement will change the data size of the column 'last_name'. It is preferable not to reduce the data size. This may result in data loss.

Adding, modifying or removing constraints will be discussed later in this section

3.5.1.4 Establishing Constraints

Following are the methods for adding constraints like primary keys, foreign keys etc.

Data type and size

It has already been mentioned as part of create table command

3.5.1.4.1 Primary key

Primary key for a table can be declared with in create table command i.e.,

```
CREATE TABLE customers
(OrderID
cust_ID         number(6)       not null  PRIMARY KEY,
first_name      varchar2(20)    not null,
last_name       varchar2(20)    not null,
city            varchar2(20),
state           char(2),
zip             char(5),
date_register   date);
```

or if the table has a composite primary key we can create the primary key as follows:

```
CREATE TABLE customers
(cust_ID        number(6)       not null,
first_name      varchar2(20)    not null,
last_name       varchar2(20)    not null,
city            varchar2(20),
state           char(2),
zip             char(5),
date_register   date,
PRIMARY KEY(cust_ID, first_name, last_name));
```

Primary key can also be declared after table has been created using alter table command i.e.,

ALTER TABLE customers ADD CONSTRAINT cust_PK PRIMARY KEY (cust_ID)

Where cust_PK is the constraint name.

3.5.1.4.2 Foreign Key (for referential integrity)

Foreign keys can also be added using CREATE TABLE or ALTER TABLE commands

```
CREATE TABLE orders
(orderID        number(5) ,
```

cust_ID number (6),

Order_date date,

Order_val number (6),

FOREIGN KEY (cust_ID) REFERENCES customers(cust_ID));

The above statement shows that in the table orders, the cust_ID attribute is a foreign key and it refers to the cust_ID attribute in customers table. This method of creating foreign keys is more useful if we are dealing with composite keys. If the primary key of the table that is being referred to is a single attribute (in our case, column cust_ID for orders table) then foreign keys can be created as follows:

CREATE TABLE orders

(orderID number(5) ,

cust_ID number (6) REFERENCES customers,

Order_date date,

order_val number (6));

In the above statement, we didn't mentioned column name for the referenced table (customers table) because the name of the foreign key attribute in both the tables is the same. In case the name differs then we have to explicitly mention column name.

Note: We can create multiple foreign keys from a single CREATE TABLE statement

foreign keys can also be created using ALTER TABLE command as

ALTER TABLE orders ADD CONSTRAINT cust_ord_FK FOREIGN KEY (Cust_ID) REFERENCES customers (Cust_ID);

Some other constraints

- CHECK
- DEFAULT
- UNIQUE

Each of the above mentioned constraints can be applied through CREATE TABLE or ALTER TABLE statements.

3.5.1.4.3 CHECK

This constraint is applied to a particular attribute to limit the range of its values e.g., there is a 'gender' attribute in a table. The only possible values are 'M' or 'F'. so the constraint can be applied as

CHECK gender IN ('M' , 'F')

CHECK orderID IN (orderID BETWEEN 1 AND 1000)

CREATE TABLE table_name

(column definitions,

CONSTRAINT chk_col CHECK colum_name IN (specific values or range of values));

In the above generic statement, column definitions means definition of attributes and 'chk_col' is user defined constraint name.

Or

CREATE TABLE table_name

(col1 data type,

col2 data type,

col3 data type CHECK IN (specific values or range of values));

similarly by using ALTER TABLE command,

ALTER TABLE table_name ADD CONSTRAINT chk_col1 CHECK attribute_name IN (specific values or range of values)

Range of the values can be specified as

CHECK order_val IN (order_val BETWEEN 0 AND 500)

This means that the order_val for a particular order should be between 0 and 500 dollars.

3.5.1.4.4 DEFAULT

This constraint will assign a default value to the attribute when a new record is inserted and no value is assigned to the subject attribute e.g.,

DEFAULT value

e.g.,

CREATE TABLE orders

(orderID number(5) ,

cust_ID number (6),

Order_date date,

Order_val number (6) DEFAULT 0);

i.e., The default order value is zero.

3.5.1.4.5 UNIQUE

Unique constraint means that the attribute values in question be unique in the table.

CREATE TABLE customers

(cust_ID number(6) not null,

first_name varchar2(20) not null,

last_name varchar2(20) not null,

city varchar2(20),

state char(2),

zip char(5),

date_register date

CONSTRAINT unq_name UNIQUE(first_name, last_name));

or

ALTER TABLE customers ADD CONSTRAINT unq_name UNIQUE (first_name, last_name)

in the above examples, customer name is defined as unique in the table customers.

3.5.1.4.6 Dropping Constraints

Constraints can be removed by using ALTER TABLE command i.e.,

ALTER TABLE table_name DROP constraint_name;

In order to get the list of all the constraints pertaining to a table, developed by a particular user etc., DBMS vendors normally use their own specific commands and statements.

3.5.1.5 Indexing

Indexing is used for better querying performance. When we create primary keys or unique constraints, unique indexes on those attributes are automatically created. We can also create indexes on non-key attributes. Suppose we want to create an index for our customers table on 'last_name' attribute then

CREATE INDEX cust_ind1 on customers (last_name);

Where cust_ind1 is the user-defined name of the index file we have created.

We can also create index on multiple attributes

CREATE INDEX cust_ind2 on customers (last_name, zip)

Indexing capabilities vary from one DBMS platform to another. For Oracle 8I, we can include as many as 16 attributes in one indexes.

Unique index can be created by adding 'UNIQUE' clause in the create statement.

CREATE UNIQUE INDEX emp_cust_ind1 ON employees (empID);

In the above example, a unique index is created for the table Employees on empID attribute.

3.5.2 Data Manipulation

3.5.2.1 Data Insertion

Data can be inserted in a table by

INSERT INTO table_name VALUES (value1, value2,....);

or

INSERT INTO table_name (attribute1, attribute2,...) VALUES (value1, value2,....);

Examples:

INSERT INTO orders VALUES (1, 10, '31-aug-200', 100);

If we don't specify particular attributes, we have to mention all the data values in order. if we don't have a value for some attributes, NULL can be specified.

INSERT INTO customers (cust_ID, first_name, last_name, date_register) VALUES (100, 'isaac', 'newton', '01-jun-2001');

(Insert a new record with data values specified only for 4 attributes. Values for rest of the attributes will be NULL. For the above case, we need to follow the order of attributes as they appear in the table but we must specify values for 'not null' and primary key attributes)

3.5.2.1.1 Data insertion for customers table

insert into customers values(2, 'kamran', 'khalid', 'philadelphia', 'PA', '09941', '30-may-1999');

insert into customers values(3, 'tom', 'tigar', 'philadelphia', 'PA', '09941', '04-june-1999');

insert into customers values(4, 'nancy', 'roberts', 'piscataway', 'NJ', '08846', '01-january-2000');

insert into customers values(5, 'micheal', 'fox', 'new york', 'NY', '07564', '14-feb-2000');

Customers Table after Data Insertion

Cust_ID	First_name	Last_name	city	state	zip	Date_register
1	isaac	newton	NULL	NULL	NULL	01-Jun-2001
2	kamran	khalid	philadelphia	PA	09941	30-may-1999
3	tom	tigar	philadelphia	PA	09941	04-june-1999
4	nancy	roberts	piscataway	NJ	08846	01-jan-2000
5	micheal	fox	new york	NY	07564	14-feb-2000

3.5.2.1.2 Data insertion in orders table

insert into orders values(12, 1, '20-jan-2001', 55);

insert into orders values(21, 2, '30-may-2001', 150);

insert into orders values(22, 2, '10-jun-2001', 50),

insert into orders values(31, 3, '03-mar-2000', 100);

insert into orders values(32, 3, '05-apr-2000', 90);

insert into orders values(41, 4, '15-jan-2001', 40);

insert into orders values(42, 4, '13-nov-2001', 145);

insert into orders values(43, 4, '18-dec-2001', 80);

Orders Table after Data Insertion

OrderID	Cust_ID	Date_Order	Order_val
12	01	20-jan-2001	55
21	02	30-may-2001	150
22	02	10-jun-2001	50
31	03	03-mar-2000	100
32	03	05-apr-2000	90
41	04	05-jan-2001	40
42	04	13-nov-2001	145
43	04	18-dec-2000	80

3.5.2.2 Querying Data

Data can be queried using SELECT statement. Generic SELECT statement is as follows:

```
SELECT      [DISTINCT]      select_list
FROM        table_names
[WHERE      search_condition]
[GROUP BY   group_by_expression]
[HAVING     search_condition]
[ORDER BY order_expression [ ASC | DESC ] ]
```

Simple select statements:

SELECT first_name AS FNAME, last_name AS LNAME FROM customers;

(Query all the customer names from the table. In this example, we have used AS clause to **define** customized column title. This is just for representation purposes)

Result

FNAME	LNAME
isaac	newton
kamran	khalid
micheal	fox
nancy	roberts
tom	tigar

SELECT * FROM customers;

(Select all the records with all the attributes in the table 'customers')

SELECT DISTINCT last_name FROM customers;

(Query unique customer last names from the table 'customers'. In absence of DISTINCT clause the query results may include repeated data as it appears in the table.)

3.5.2.2.1 Applying Conditions to Queries (Use of WHERE clause):

SELECT cust_id, last_name FROM customers WHERE state ='PA';

(query last names of all the customers from Pennsylvania)

Result

Cust_ID	last_name
2	khalid
3	tigar

3.5.2.2.2 Conditions on strings (use of LIKE clause)

LIKE is used for string pattern matching. By using LIKE clause, we can apply conditions in more flexible manner i.e., we can query data starting, ending or having a particular character or sub-string.

LIKE can be used with wild card characters. Some of them are as follows:

%	Any string of zero or more characters	WHERE LIKE '%mid_str%' (strings having 'mid_str' , in the beginning, end or middle)
_ (underscore)	Any single character	WHERE LIKE '_ _end' (strings ending with 'end' and having two characters before that)

.

SELECT last_name FROM customers WHERE first_name LIKE 'k%' AND date_register BETWEEN '01-JAN-1998' AND '31-dec-2000';

(query last names of all the customers having first name that starts from 'k' and who got registered between jan 1st 1998 and dec 31st 2000)

3.5.2.2.3 Aggregate Functions

Following are the aggregate functions in SQL i.e.,

COUNT(), SUM(), MAX(), MIN(), AVG()

In case of COUNT function, null values are also counted but if we declare

COUNT (DISTINCT expression) then null values are ignored

e.g.,

SELECT COUNT(state) FROM customers

Result : 4

There are only three states i.e., NY, NJ and PA mentioned in the customers table. Isaac Newton has NULL value for state. That null value is also included in the result.

SELECT COUNT(DISTINCT state) FROM customers

Result: 3

3.5.2.2.4 Retrieving Data in Groups

GROUP BY clause helps in getting the results in groups. It is exclusively used with aggregate functions i.e., min(), max(), sum(), count() and avg(). For instance, there is a table containing information about sales at different stores of a company and we want to query the total sales at each store. In this case, we can use sum(income) function with GROUP BY 'individual store'. We can also group results by two or more attributes.

Some examples:

SELECT state, COUNT(*) AS NUMofCUST from customers GROUP BY state;

(Query the number of customers from each state)

Result:

State	NUMofCUST
NULL	1
NJ	1
NY	1
PA	2

SELECT C.cust_id, COUNT(*) AS Num, SUM(O.order_val) AS Amount FROM customers C, orders O WHERE C.cust_ID = O.cust_ID GROUP BY C.cust_id;

(List down number of orders made by each customer along with the total sum of order value)

Result:

Cust_ID	Num	Amount
1	1	55
2	2	200
3	2	190
4	3	265

Grouping can be done by more than one attribute i.e.,

SELECT state, zip, COUNT(*) AS NUMofCUST from customers GROUP BY state, zip;

All the attributes that appear in the SELECT clause with the aggregate function must appear in the GROUP BY clause e.g., in the above case, we can't just group by state or zip.

3.5.2.2.5 Applying Conditions on the Groups (Use of HAVING clause)

Having is used to exclude unwanted groups in the query result. It can only be used with GROUP BY and is applied to grouped data. The difference between WHERE and HAVING is that WHERE is used to filter out the data before 'Grouping' and HAVING' clause is used to apply condition on grouped data e.g.,

SELECT COUNT (*), state from customers WHERE last_name LIKE 'r%' GROUP BY state HAVING COUNT (*) > 10;

(List the numbers of customers by state having names starting with 'r' and include only those groups that have number of customers greater than 10)

3.5.2.2.6 Sorted Retrievals (use of ORDER BY clause)

Query results can be obtained sorted with respect to one or more attributes (more than one attribute can be used for sorting in case there is room for multi-layered sorting) by using ORDER BY clause

SELECT first_name, last_name FROM customers ORDER BY last_name;

First_name	last_name
Micheal	fox
Kamran	khalid
Isaac	newton
Nancy	roberts
Tom	tigar

SELECT state, zip, COUNT(*) AS NUMofCUST from customers GROUP BY state, zip ORDER BY state;

(group the result by state and zip and sort each group by state)

3.5.2.2.7 Joins

Related tables can be combined by some common attributes (foreign keys) to get a concatenated relation.

3.5.2.2.7.1 Natural Join (Inner Join)

Tables are joined on certain common attributes (foreign keys) and only those tuples are included in the result that satisfy the join condition mentioned in WHERE clause.

e.g.,

Query the customers with their orders

SELECT C.first_name, C.last_name, orderID, order_val FROM customers C, orders O WHERE C.cust_ID = O.cust_ID

Result:

First_name	last_name	order_id	order_val
Kamran	khalid	21	150
Kamran	khalid	22	50
Tom	tigar	31	100
Tom	tigar	32	90
Nancy	roberts	41	40
Nancy	roberts	42	145
Nancy	roberts	43	80

Here the tables are joined on the common attribute ‘cust_ID’. In this example, we defined aliases for both the tables i.e., C for customers and O for orders in order to avoid writing long table names again and again. The attributes can be referenced as

Table-name-or-alias**.**Column-name

3.5.2.2.7.2 Outer Joins

Outer joins return all the rows from at least one of the tables or views mentioned in the FROM clause, as long as those rows meet join conditions. For left outer join, all rows are retrieved from the left table referenced in the statement and all rows from the right table referenced in a right outer join. All rows from both tables are returned in a full outer join.

SELECT col_names FROM tab1 LEFT OUTER JOIN tab2 ON common_attribute

(In this case, all the rows from tab1 will be included in the output but only those rows from the tab2 which satisfy the conditions)

SELECT col_names FROM tab1 RIGHT OUTER JOIN tab2 ON common_attribute

(In this case, all the rows from tab2 will be included in the output but only those rows from the tab1 which satisfy the conditions)

SELECT col_names FROM tab1 FULL OUTER JOIN tab2 ON common_attribute

(In the above example, all the rows from tboth the tables will be included in the output)

for example,

SELECT C.first_name, C.last_name, orderID, order_val FROM customers C FULL OUTER JOIN orders O ON C.cust_ID = O.cust_ID

Result:

Kamran	khalid	22	50
Tom	tigar	31	100
Tom	tigar	32	90
Nancy	roberts	41	40
Nancy	roberts	42	145
Nancy	roberts	43	80
Micheal	fox	NULL	NULL

We can see that in the above case, although Michael fox has no record matching the join condition but still it is appearing in the result and it is because of full outer join.

3.5.2.2.8 Set Operations as applied to Tables

In order to apply these operations on tables, both the tables/result sets (query results) should have same attributes defined in the same order. The relations resulting from these operations are sets of tuples i.e., duplicate tuples are eliminated from the result.

3.5.2.2.8.1 Union

Union of two relations will be a aggregated relation containing all the tuples present in the two tables e.g.,

Query all the customers who made orders in year 2000 and those who ordered in year 2001 and they belong to PA state.

(SELECT cust_ID FROM customers Cus, orders Ord WHERE Cus.Cust_ID=Ord.Cust_ID

AND YEAR (order_date) = '2000') UNION (SELECT cust_ID FROM customers Cus, orders Ord WHERE Cus.Cust_ID=Ord.Cust_ID

AND YEAR (order_date) = '2001' AND Cus.state ='PA')

Result:

Cust_ID

2

3

4

3.5.2.2.8.2 Intersection

Intersection operation will generate common attribute values common in both data sets

Following is a simple generic intersection operation:

(SELECT col1, col2 FROM tab1, tab2 WHERE join and other conditions)

INTERSECT

(SELECT col1, col2 FROM tab3, tab2 WHERE join and other conditions)

3.5.2.2.8.3 Difference

Difference operation will return a resultset having those values that are present in the first set (first sub query) but not in the 2nd set (2nd sub-query). It is implemented by EXCEPT clause i.e.,

(SELECT col1, col2 FROM tab1, tab2 WHERE join and other conditions)

EXCEPT

(SELECT col1, col2 FROM tab3, tab2 WHERE join and other conditions)

3.5.2.2.9 IN, NOT IN, ANY and ALL Clauses

These operators are used in WHERE clause. IN and NOT IN clauses are work in the same way as check constraint i.e., they are used to restrict the search to certain set of values or to exclude certain set of values ie.,

SELECT col1 FROM customers WHERE col2 IN (set of values or sub-query)

Suppose we have added another column in the orders table viz. pay_satus which can have only three values i.e., 'P' (dues paid), 'N' (Not paid), and 'D' (declared defaulted payment). As we have discussed earlier, this constraint can be enforced by a CHECK clause.

SELECT order_id FROM orders WHERE pay_status IN ('N', 'D');

(Select the orders for which payment has not been made or has been declared defaulted.)

SELECT order_id FROM orders WHERE pay_status NOT IN ('N', 'D');

(Select the orders for which payment has already been made.)

SELECT last_name FROM customers WHERE ID IN (SELECT SUM (order_val), id FROM orders GROUP BY ID HAVING SUM (order_val) > 500);

ANY and ALL are used to compare values with a single column set of values.

SELECT col FROM table_name WHERE col > ANY (set of values of same data type or a sub-query returning a set of values) ;

(Query all 'col' attribute from the table where every col value is greater than at least one value in the set mentioned after ANY clause)

SELECT col FROM table_name WHERE col > ALL (set of values of same data type or a sub-query returning a set of values) ;

(Query all 'col' attribute from the table where every col value is greater than all the values in the set mentioned after ALL clause)

Note:

ALL and ANY can be usd with following comparison operators

= , > , > = , <, < =,

! > (not greater than) ,

! < (not greater than),

!= , <> (not equal to)

3.5.2.3 Data Modification

3.5.2.3.1 Deletion

DELETE FROM table_name WHERE attribute_name = some_value

Examples:

DELETE FROM customers

(Delete all the tuples in the customers table without disturbing the table structure)

DELETE FROM customers C WHERE C.ID > 100

(Delete all the customer records whose id is greater than 100)

record deletion involving nested query

DELETE FROM customers where id IN (SELECT id FROM orders O, customers C WHERE C.ID = O.ID and order_date > '31-dec-2000')

(Delete all the customer records who ordered after 31st december 2000)

3.5.2.3.2 Update

UPDATE table_name SET column_name = new_value WHERE column_name = some_value

Examples:

UPDATE customers SET zip = '08846' WHERE last_name = 'khalid'

(Change the zip code of all the customers to 08846 whose last name is 'khalid')

UPDATE customers SET register_date = '31-dec-2001'

(Update registration date for all the tuples in customers table)

3.5.3 Data Administration

3.5.3.1 Views

A view as mentioned in the previous section is a virtual table that is derived from other tables .

Views can be created by CREATE VIEW statement. Syntax for view creating statement is

CREATE VIEW view_title AS (query for which view is desired)

Examples:

Create a view on the first_name, last_name and zip columns of customers table.

CREATE VIEW view1 AS (SELECT last_name, first_name, zip, state FROM customers)

View can also be created on table joins.

Once created, we can query a view just like an ordinary table.

3.5.3.1.1 Administrative Functions

3.5.3.1.1.1 Database

New database can be created by

CREATE SCHEMA store AUTHORIZATION mary

(CREATE SCHEMA is used by oracle databases. For SQL Server databases, CREATE DATABASE command is used)

In the above statement, a database named 'store' is created and table creation privileges has been granted to the user 'mary'

Now whatever tables the user 'mary' will create in the database 'store' will be owned by her and she could grant access on her tables to other users.

3.5.3.1.1.2 Allocating and revoking permissions

Create

A new user can be created by

CREATE USER username IDENTIFIED BY password

Grant

Access to certain tables or views can be granted to different users using GRANT command i.e.,

GRANT privileges ON table or view names TO user [WITH GRANT OPTION]

In the above generic GRANT statement, privileges can be

SELECT (querying privilege), UPDATE (data modification privilege) , DELET, INSERT etc.

WITH GRANT OPTION clause authorizes the users to further grant same permission or privilege to other users.

e.g.,

Grant querying and modification privileges to all users

GRANT SELECT, UPDATE ON customers, orders TO PUBLIC

Grant database administrator privilege to user 'bob'

GRANT dba TO bob

Revoke

The database administrator can revoke granted privileges by

REVOKE privileges ON table or view names FROM user name

e.g.,

revoke update permission from the user 'bob'.

REVOKE UPDATE ON orders FROM bob

Note: the users should consult the dbms manual or help files to know the exact syntax of statements and commands. Here we have used the commands that are typically used for Oracle databases

3.5.4 References for Additional Information

- Elmasri et al, "Fundamentals of database systems", Addison-Wesley, 2001
- www.microsoft.com/msdn (Microsoft Developer's Network)

3.6 Object-Oriented Systems and Databases

3.6.1 Introduction

Relational databases are suitable for many applications. However, it is not easy to represent complex information in terms of relational tables. For example, a car design, a computing network layout, and software design of an airline reservation system cannot be represented easily in terms of tables. For these cases, we need to represent complex interrelationships between data elements, retrieve several versions of design, represent the semantics (meaning) of relationships, and utilize the concepts of similarities to reduced redundancies.

The next generation of database systems, commonly known as the object-oriented database management systems (OODBMSs), have been developed to support applications in computer aided design and computer-aided manufacturing (CAD/CAM), expert systems, computer-aided software engineering (CASE), and office automation. Simply stated, OODBMSs combine and extend the features of database management systems, artificial intelligence, and "object oriented programming" for these and other future applications.

It seems that we are in the middle of an object-oriented "revolution". There is object-oriented analysis, object-oriented design, object-oriented programming, object-oriented databases, and so on. Due to too many object-oriented "things", many groups are trying to figure out what to do. An example is the Object Management Group (OMG) which has been formed as a non-profit consortium of several hundred software and systems manufacturers and technology information providers. OMG has developed, among other things, CORBA (Common Object Request Broker Architecture) for distributed applications and UML (Universal Modeling Language) to aid application development. Information about OMG can be found at their web site (www.omg.org).

For our purpose, we focus on OODBMS. However, many OODBMSs are very closely related to, or are extensions of, object-oriented programming languages. For a quick overview of object oriented concepts, see the chapter "Object Oriented Concepts -- A Short Tutorial" in this module. .

3.6.2 Object-Oriented Databases

Object-oriented databases allow storage and retrieval of non traditional data types such as bitmaps, icons, text, polygons, sets, arrays and lists. The objects can be simple or complex, can be related to each other through complex relationships, and can inherit properties from other objects. Object-oriented database management systems (OODBMS), which can store, retrieve and manipulate objects, have been an area of active research and exploration since the mid 1980s. Most of the work in OODBMSs has been driven by the computer aided design and computer aided manufacturing (CAD/CAM) applications.

OODBMS allow complex relationships between data entities. They combine several features of DBMS, AI and software engineering. In an OODBMS, the data is viewed as objects with the following stipulations:

- Simple object is a relational table
- Complex objects are built from simple objects
- Each object has properties (attributes)
- Objects are inter-related through complex relationships
- Relationships may carry semantics (meaning). This primarily involves two types of relationships: inheritance (is-a and a-kind-of –ako) and aggregation (part-of and contains)
- DDL may use inheritance to create new objects
- The DML can be customized with a general query language that uses AI pattern matching
- You can store procedures with the data

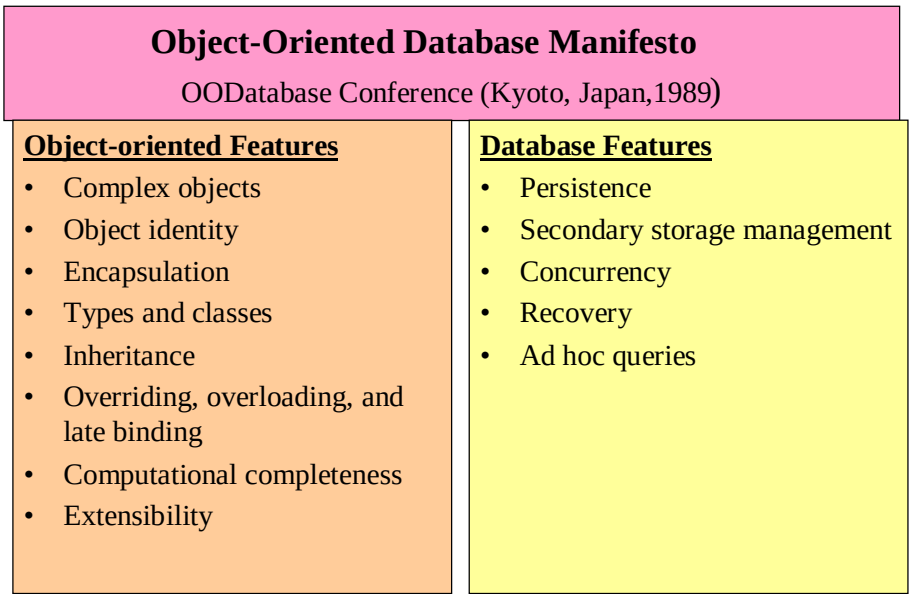


Figure 3-9: The Object Oriented Database Manifesto

What exactly is an OODBMS? This question has been asked since the mid 1980s. In 1989, a group of computer scientists got together and established "The Object-Oriented Database Manifesto" [Atkinson 1989]. This Manifesto, displayed in Figure 3-9, establishes the basic properties of OODBMS by combining conventional database functionalities with object-oriented functionalities. According o this manifesto, the key properties of OODBMSs are (for an expanded discussion of the manifesto, see the chapter "Object Oriented Concepts -- A Short Tutorial" in this module):

- Data may be stored, retrieved and manipulated as complex objects which consist of sets, lists, arrays or relational tuples.
- OODBMSs allow creation of objects from existing objects by using inheritance of properties.
- Procedures can be stored as objects in the database.
- The relationships between objects can be complex, many to many relationships.
- Objects can contain very large values to store pictures, voice or text.
- Most OODBMSs provide facilities to track multiple versions of an object.
- In many OODBMS, the data manipulation language is closely related to the programming language

The OODBMS systems generally fall into two categories:

- Extensions of the object-oriented programming languages (OOPL) to include the features of DBMS. Examples are O2 and Gemstone systems.
- Extensions of the relational DBMS to include the features of OOPL. Examples are Starburst and POSTGRESS.

OODBMS have moved from state of the art to state of the market, however they are not heavily used at the time of this writing (less than 5% of corporate data is stored in OODBMS). Objectstore is one of the most popular OODSBMS. It is beyond the scope of this book to discuss detailed features of existing OODBMS.

Advantages/Disadvantages of OODBMSs. OODBMSs have emerged due to the limitations of relational DBMSs in handling complex relationships and semantics. In addition, OODBMS attempt to include desirable features from AI and software engineering to improve application reusability and maintainability. Despite several potential advantages of OODBMSs, a few concerns should be noted.

First, standard query languages for OODBMS such as OQL are not very popular commercially. In addition, the performance characteristics of OODBMSs are not well understood. This problem is expected to be addressed by performance improvements in the OODBMSs. We have to see how OODBMSs operate in large applications with thousands of users.

3.6.3 Objectizing a RDBMS

Although OODBMSs have not been very successful commercially, handling objects in EB applications is to handle the OO views on relational databases. The main motivation is that most programs work on objects and should not have to translate between object and relational views. Basically, RDBMS operations need to be performed on objects (i.e., fileds), on sets of objects (i.e., rows), and on tables (i.e., joins). The different approaches to "objectizing RDBMSs" are:

- BLOB (binary large objects) support in RDBMS. Blobs allow you to store, retrieve, and display graphics, memos, and video clips. However, BLOBS do not allow querying the content of the BLOB (i.e., you can store and display the pictures of employees but cannot select the employees who have images).
- View relational data as a collection of methods provided by the RDBMS. For example, the SQL statements such as create, select, update, insert, delete, and grant can be thought of as methods that are performed on the data.
- Build wrappers around SQL (i.e., treat each SQL statement as an OO statement). The wrapper contains classes to create a table, to insert a record, to read arecord, to examine the satus of an object, and to do do several other low level database actions. Warappers of this type are becoming commercially available (e.g., DBTools.h++ from Roguewave). This is not a problem in static SQL because in this case, the variablesa are bound at startup time and type checking is done at compile time. However, for dynamic SQL, this is tough to do because how can you know about variables and type checking.
- Build wrappers around the database itself (i.e., make tables look like a set of objects and perform operations on them). These wrappers are more sophisticated and map application objects to relational tables (some wrappers provide the reverse functionality also). These wrappers support encapsulation (i.e., methods to define the inetrfaces to the databases, inheritance (i.e., the sharing of attributes, queries, methods, and relationships between objects); and associations (i.e., the relational foreign key relationships are associated to relationships among classes). These wrappers may also map relational database capabilities to object classes. And support object caching, transactions, and database efficiency. The wrappers provided by Persistence Software fall into this category.

3.7 Overview of Database Design

Database design attempts to provide consistent and current information to the end users in a speedy fashion. The design process goes through the following general steps, illustrated through a simple example in Figure 3-10:

1). Capture Information Requirements. The information requirements of different users and applications are developed after interviewing different sets of users.

2).Build a Logical Data Model (LDM). An LDM represents user data requirements and contains the following pieces of information:

- Entities (objects) such as customers, parts, products, students
- Attributes of entities such as the name and address of customer
- Relationships between objects such as customers buy products.

Different people may develop different views of LDM (user view, management view, programmer view). These views are usually integrated to create a common corporate LDM. In addition, the LDM is cleaned up to remove synonyms, homonyms, and derived data. The resultant LDM can be represented as an ERA diagram.

3). Develop a Database Design. This step first creates a "normalized" database structure. Normalization is a procedure for decomposing large data entities to remove update anamoly (should not have to update more than is needed) and delete anamoly (should not delete more than is needed). Normalization, not needed for retrieval only data, should be in 1^{st}, 2^{nd} and 3^{rd} normal forms. Discussion of normalization is beyond the scope of this book. See the side bar "An Informal Normalization Example" for the key ideas.

After normalization, known as logical database design, a physical database design translates the normalized logical data model into a DBMS supported physical structure. The physical design depends on the DBMS type. In case of a relational database system, the physical design shows the tables, the attributes of each table, and the indexes (see section C3). Detailed discussion of this topic is also beyond the scope of this book.

In a distributed environment, the design also includes the following steps.

4). Data Partitioning and Clustering. This step attempts to decompose data for application/user specific requirements. For example, as shown in Figure 3-10, the customer table is partitioned into two regions because the customers live in two regions. The partitions may also be clustered because some information is always used together.

5). Data Allocation. The clustered data is allocated to different physical sites to minimize response time and improve availaability. A very large number of data allocation algorithms have been developed in the academic community under the heading of "file allocation problem (FAP)".

The book "Database Design" by Toby Teorey (Prentice Hall) is an excellent source of information on this topic.

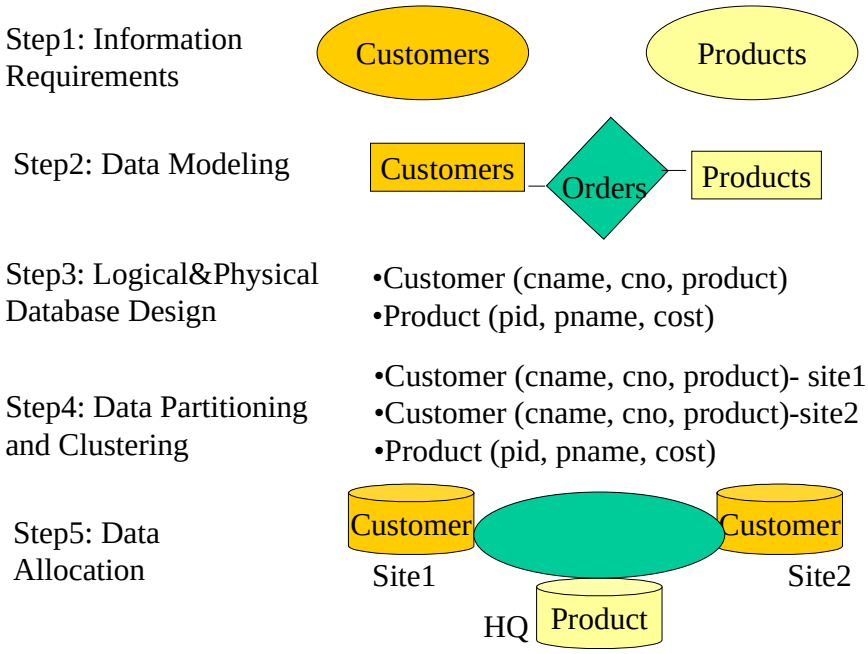


Figure 3-10: A Database Life Cycle

An Informal Database Normalization Example

Let us start with a table that represents a company (the underlined attributes indicate primary keys):

Company (ename, ssn, bdate, address, dnumber, dname, dmanager, pnumber, hours, pname, plocation)

This table is too large, we may want to subdivide into two:
Emp-Dept (ename, ssn, bdate, address, dnumber, dname, dmanager)
Emp-Project (ssn, pnumber, hours, ename, pname, plocation)

Normalization is basically a formal method to decompose: the tables.
- First normal form: a table with fixed attributes and no duplicate records
- Second normal form: every non-primary attribute must depend fully on the primary key

Let us consider the table:
Emp-Project (ssn, pnumber, hours, ename, pname, plocation)

For 2nd normal form, break into two tables: T1(ssn, pnmumber, hours), T2(ssn, ename), T3(pnumber, pname, plocation). Now let us look at the third normal form:

- Third normal form: non-prime attributes must not depend on each other (no subtables)

Let us now consider the table

Emp-Dept (ename, ssn, bdate, address, dnumber, dname, dmanager)

To make it in 3rd normal, split into two; T1 (ename, ssn, bdate, address) and T2(dnumber, dname, dmanager)

The result of normalization is the following 5 tables generated from the company table: :

Employee (ename, ssn, bdate, address, dnumber)
Department (dname, dnumber, dmanager, dlocation)
Dept-location (dnumber, dlocation)
Project (pname, pnumber, plocation, dnumber)
Works_on (ssn, pnumber, hours)

Keep in mind that too much normalization produces large number of tables that may create performance problems due to too many joins. Denormalization consolidates tables back for performance improvement.

3.8 Chapter Summary

Different databases are used for different applications in a distributed computing environment. As stated previously, relational database technology is state of the market and state of the practice. SQL, the query language for relational DBMS has become a de-facto standard for enterprise-wide data access, even for non-relational data sources. However, relational DBMSs are not suitable for many engineering and other emerging applications discussed in the previous section. Object-oriented DBMSs are state of the market but not heavily state of the practice at the time of this writing.

The main question is which database technologies are suitable for which applications? Fig. E.12 attempts to answer this question by using the data and process complexity of applications as a measure. For example, the x-axis shows the complexity of the data model (one to one versus many to many relationships) and the y-axis shows the complexity of the processes (simple retrieval and storage versus complex computations). This figure, based on a diagram produced by Ontologic Corp., shows the regions where some of the database technologies can be most effective. For example, it shows that the older network and hierarchical DBMSs are more suited for complex data but relatively simple processing type applications while the relational DBMSs are more suitable for applications with relatively simple data models. Theoretically, OODBMSs are intended for the complex data and complex processing type applications. A few applications are shown in Fig. E.12 for illustrative purposes.

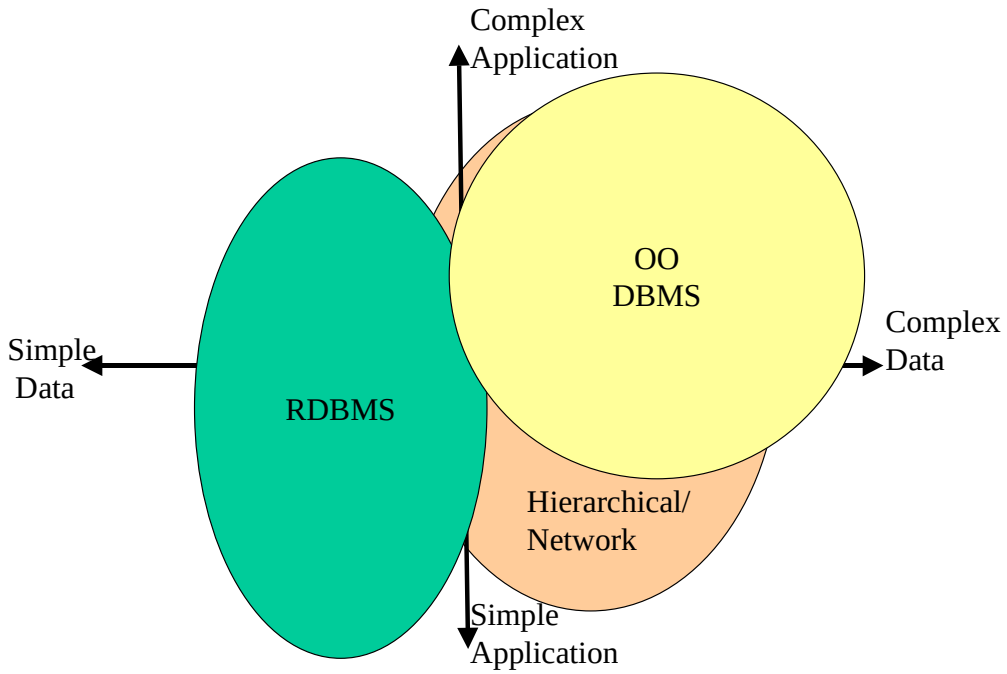


Figure 3-11: A Model for Evaluating Database Technologies

3.9 CASE STUDY: Databases for XYZCORP

XYZCORP has embarked on a corporate-wide database effort. As a result of this effort, you have been assigned the following three projects.

Project A): Define the databases needed for the IMCS (Integrated Manufacturing Control System) project described in chapter 8 Case Study. For each database, you should:

- Define the logical data model for IMCS
- Define the main attributes of the database in IMCS
- Describe at least 5 queries, in English, against the database
- Choose an appropriate data model

Project B): Assume that the following databases have been defined for the stores and the products sold in different stores in IMCS:

. STORE (S-NAME, S-ID, S-ADDRESS, S-MANAGER, PROD-ID) . PRODUCT (PROD-ID, PROD-NAME, PROD-TYPE, PROD-DESCRIPTION)

- Create these two databases by using SQL
- Translate the following simple queries to SQL: .

  List the store names in Michigan .

  List the names and ids of product type = "PC" .

  List the names of the stores that carry product type = "PC" .

  Count the total number of stores .

  Count the total number of distinct products .

  Delete the product type "radio" from the products

Create indices on PROD-ID and S-ID

Project C): Design a Bill of Materials (BOM) database for the IBM PC compatible desktop computers. You must include all materials (connectors, adapters) in this database. Create the BOM database by using SQL statements and issue at least 5 SQL queries against the BOM database. What are the limitations of the relational data model, if any, for this database.

The administration has also asked you to create a database which describes the XYZCORP network. The backbone network used by XYZCORP has been described in other chapters. This database will be used for different purposes: to maintain an inventory of the network devices, to provide information for network management and to support expert systems. List the main attributes of this database and choose an appropriate data model for this datanbase**.**

Hints about the Case Study

Project A). The LDM for IMCS would show business entities (customers, bills, inventory, etc), engineering entities (designs, test results, etc) and manufacturng entities (robots, cells,e tc). Many of these entities can be represented as relational or object-oriented databases.

Project B). This is a straightforward application of SQL

Project C). The BOM database can be a relational database. However, the network configuration database should be an object-oriented database. You should especially view this database as a knowledgebase for many tools in XYZCORP.

3.10 Key References

Atkinson, M., et al, "The Object-Oriented Database Manifesto", Proceedings of the International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan, Dec. 1989.

Bernstein, P.A. and Chu, D.W., "Using Semi_joins to Solve Relational Queries", Journal of ACM, Jan. 1981.

Bobak, A., "Data Modeling and Design for Today's Architectures", Artech House, 1997

Booch, G., "Object Oriented Design with Applications", Addison-Wesley, 1994 (second edition).

Codd, E.F., "Fatal Flaws in SQL", Datamation, August 15 and September 1, 1988.

Date, C.J., "An Introduction to Database Systems", Fifth Edition, Vol. 1 and Vol. 2, Addison-Wesley, 1990.

Date, C.J., "A Guide to DB2", Addison Wesley, 1984

Date, D.J., "A Guide to The SQL Standard", Addison-Wesley, 1987

Dowgiallo, E., "An Introduction to SQL", Micro Systems, September 1988.

Elmasri, R. and Navathe, S., "Fundamentals of Database Systems", Benjamin-Cummings, 2001

Gray, J., "The Transaction Concept: Virtues and Limitations", Proceedings of Conference on Very Large Databases, Sept. 1981, pp. 144-154.

Hardwick, M., and D.L. Spooner, "Comparison of Some Data Models for Engineering Objects", IEEE CG&A, March 1987

Hernandez, M. "Database Design for Mere Mortals : A Hands-On Guide to Relational Databases" Addison-Wesley, 1997
Hursch, C. and Hursch, J., "SQL: The Structured Query Language", TAB Books, 1988

Inmon, W., "Optimizing Performance with Denormalization", Database Programming and Design, Premier Issue, 1987

Muller, J., "Database Design for Smarties: Using UML for Data Modeling", Morgan Kaufman, 1999

Purba, S (editor)., "Data Management Handbook", 3rd Edition, Auerbach, 1999

"Schur, S. "The Database Factory : Active Database for Enterprise Computing", John Wiley, 1994

Stroustrup, B., "The C++ Programming Language", Addison-Wesley, 1986

Teorey, T.J and Fry, J.P "Design of Database Structures", Prentice Hall, 1982.

Owens, K., " Building Intelligent Databases With Oracle Pl/Sql, Triggers, and Stored Procedures", (2nd Edition), Prentice Hall, 1998.

Vinzant, D., "SQL Database Servers", Data Communications, January 1990, pp. 72-88.