**CHAPTER FIVE**

**SQL Relational Database**

**Introduction**

This chapter presents the main features of the SQL standard for commercial DBMSs

Upon completion of this Chapter you will be able to:

.

**Terminology**

**Relation:** Table

**Tuple:** Row or Record

**Relational Data Structure**

**Relation:** A relation corresponds to a table

**Tuple:** Corresponds to a row of a table

**Domain:** Is a pool of values from which one or 2 values draw their actual values e.g. the Town Domain is a set of all legal town names. A relation on domains D1, D2, D3........,Dn (not necessarily all distinct) consists of a heading and a body, the heading consists of a fixed head of attributes a1, a2, a3, ........... an such that each attribute ai corresponds to exactly one of the underlying domain Di. The body consists of a time varying set of tuples where each tuple in turn consists of a set of attribute value pairs (ai, ri)

**Background on SQL**

Conceived in IBM San Jose research lab in the mid-70s as a database language for the new relational database model proposed by Edgar F. Codd, an IBM researcher.

SQL commands fall into three main categories: DDL, DML and

DCL.

DDL: create, alter, drop

DML: select, insert, update, delete

DCL: commit, rollback, grant, revoke

SQL-92 also included host language bindings (C, ADA). Had compliance levels: Entry, Transitional, Intermediate, Full. Almost no DBMS vendor complies fully. Just like they don’t comply with Codd’s 12 guidelines for RDBMSs.



Goal of SQL3: Turn SQL-92 into a complete language for the definition and management of persistent, complex objects. Therefore, it includes constructs for

• Generalization and specialization hierarchies

• Multiple inheritance

• User defined and abstract data types (ADTs)

• Triggers

• Assertions

• Support for knowledge-based systems

• Support for knowledge-based systems

• Recursive query expressions

• Additional data administration tools

• Object identifiers

• Inheritance

• Polymorphism

• Encapsulation

• Others associated with object data management

**Data Definition in SQL**

Used to CREATE, DROP, and ALTER the descriptions of the tables

(relations) of a database

**i. CREATE TABLE:**

- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL (i,j), CHAR(n), VARCHAR(n))

- A constraint NOT NULL may be specified on an attribute

CREATE TABLE DEPARTMENT

( DNAME VARCHAR(10)NOT NULL, DNUMBER INTEGER NOT NULL, MGRSSN CHAR(9), MGRSTARTDATE CHAR(9) );



- In SQL2, one can use the CREATE TABLE command for specifying the primary key attributes, secondary keys, and referential integrity constraints (foreign keys)

- Key attributes can be specified via the PRIMARY KEY and UNIQUE

phrases

CREATE TABLE DEPT

( DNAME VARCHAR(10) NOT NULL, DNUMBER INTEGER NOT NULL, MGRSSN CHAR(9), MGRSTARTDATE CHAR(9), PRIMARY KEY (DNUMBER), UNIQUE (DNAME),

FOREIGN KEY (MGRSSN) REFERENCES EMP );

**ii. DROP TABLE:**

- Used to remove a relation (base table) *and its definition*

- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists

Example:

DROP TABLE DEPENDENT;

**iii. ALTER TABLE:**

**-** Used to add an attribute to one of the base relations

- The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is *not allowed* for such an attribute

Example:

ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);

- The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple. This can be done using the UPDATE command.

**iv. CREATE SCHEMA:**

- Specifies a new database schema by giving it a name

CREATE SCHEMA COMPANY;

**Referential Integrity options**

- In SQL2, we can specify CASCADE or SET NULL or SET DEFAULT

on referential integrity constraints (foreign keys)

CREATE TABLE DEPT

(DNAME VARCHAR(10) NOT NULL, DNUMBER INTEGER NOT NULL, MGRSSN CHAR(9), MGRSTARTDATE CHAR(9), PRIMARY KEY (DNUMBER), UNIQUE (DNAME),

FOREIGN KEY (MGRSSN) REFERENCES EMP

ON DELETE SET DEFAULT ON UPDATE CASCADE );

CREATE TABLE EMP

(ENAME VARCHAR(30) NOT NULL, ESSN CHAR(9),

BDATE DATE,

DNO INTEGER DEFAULT 1, SUPERSSN CHAR(9), PRIMARY KEY (ESSN),

FOREIGN KEY (DNO) REFERENCES DEPT ON DELETE SET DEFAULT ON UPDATE CASCADE,

FOREIGN KEY (SUPERSSN) REFERENCES EMP ON DELETE SET NULL ON UPDATE CASCADE );

- Many other features, e.g., joined relations (all not likely to be all implemented)

**Data Types in SQL**

- Has DATE, TIME, and TIMESTAMP data types

**DATE:**

- Made up of year-month-day in the format yyyy-mm-dd

**TIME:**

- Made up of hour:minute:second in the format hh:mm:ss

**TIME(i):**

- Made up of hour:minute:second plus i additional digits specifying fractions of a second

- format is hh:mm:ss:ii...i

**TIMESTAMP:**

- Has both DATE and TIME components

**INTERVAL:**

- Specifies a relative value rather than an absolute value

- Can be DAY/TIME intervals or YEAR/MONTH intervals

- Can be positive or negative

- When added to or subtracted from an absolute value, the result is an absolute value

**Retrieval Queries in SQL**

SQL has one basic statement for retrieving information from a database;

the SELECT statement

- This is *not the same as* the SELECT operation of the relational algebra

- Important distinction between SQL and the formal relational model; SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values

- Hence, an SQL relation (table) is a *multi-set* (sometimes called a bag) of tuples; it *is not* a set of tuples

- SQL relations can be constrained to be sets by specifying PRIMARY KEY or UNIQUE attributes, or by using the DISTINCT option in a query

- Basic form of the SQL SELECT statement is called a *mapping* or a

*SELECT-FROM-WHERE block* **SELECT** <attribute list> **FROM** <table list> **WHERE** <condition>

o <attribute list> is a list of attribute names whose values are to be retrieved by the query

o <table list> is a list of the relation names required to process the query o <condition> is a conditional (Boolean) expression that identifies the

tuples to be retrieved by the query

**Simple SQL Queries**

- Basic SQL queries correspond to using the SELECT, PROJECT, and

JOIN operations of the relational algebra

- All subsequent examples use the COMPANY database

- Example of a simple query on *one* relation

Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

**Q0:** SELECT BDATE, ADDRESS FROM EMPLOYEE

WHERE FNAME='John' AND MINIT='B' AND LNAME='Smith'

- Similar to a SELECT-PROJECT pair of relational algebra operations; the SELECT-clause specifies the *projection attributes* and the WHERE-clause specifies the *selection condition*

- However, the result of the query *may contain* duplicate tuples

Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

**Q1:** SELECT FNAME, LNAME, ADDRESS FROM EMPLOYEE, DEPARTMENT



WHERE DNAME='Research' AND DNUMBER=DNO

- Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations

- (DNAME='Research') is a *selection condition* (corresponds to a

SELECT operation in relational algebra)

- (DNUMBER=DNO) is a *join condition* (corresponds to a JOIN

operation in relational algebra)

Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

**Q2:** SELECT PNUMBER, DNUM, L NAME, BDATE, ADDRESS FROM PROJECT, DEPARTMENT, EMPLOYEE

WHERE DNUM=DNUMBER AND MGRSSN=SSN AND PLOCATION='Stafford'

- In Q2, there are *two* join conditions

- The join condition DNUM=DNUMBER relates a project to its controlling department

- The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department

**Aliases, \* and DISTINCT, Empty WHERE-clause**

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in *different relations*

- A query that refers to two or more attributes with the same name must *qualify* the attribute name with the relation name by *prefixing* the relation name to the attribute name

- Example: EMPLOYEE.LNAME, DEPARTMENT.DNAME

**ALIASES:**

- Some queries need to refer to the same relation twice

In this case, *aliases* are given to the relation name

Query 8: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.



**Q8:** SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME FROM EMPLOYEE E S

WHERE E.SUPERSSN=S.SSN

- In Q8, the alternate relation names E and S are called *aliases* for the

EMPLOYEE relation

- We can think of E and S as two *different copies* of the EMPLOYEE relation; E represents employees in the role of *supervisees* and S represents employees in the role of *supervisors*

- Aliasing can also be used in any SQL query for convenience

- Can also use the AS keyword to specify aliases

**Q8:** SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME FROM EMPLOYEE AS E, EMPLOYEE AS S WHERE E.SUPERSSN=S.SSN

**UNSPECIFIED WHERE-clause:**

- A *missing WHERE-clause* indicates no condition; hence, *all tuples* of the relations in the FROM-clause are selected

- This is equivalent to the condition WHERE TRUE Query 9: Retrieve the SSN values for all employees.

**Q9:** SELECT SSN

FROM EMPLOYEE

- If more than one relation is specified in the FROM-clause *and* there is no join condition, then the *CARTESIAN PRODUCT* of tuples is selected

Example:

**Q10:** SELECT SSN, DNAME

FROM EMPLOYEE, DEPARTMENT

- It is extremely important not to overlook specifying any selection and join conditions in the WHERE-clause; otherwise, incorrect and very large relations may result

**USE OF \*:**

- To retrieve all the attribute values of the selected tuples, a \* is used, which stands for *all the attributes*

Examples:

|  |  |  |
| --- | --- | --- |
| **Q1C:** | SELECT | \* |
|  | FROM | EMPLOYEE |
|  | WHERE | DNO=5 |

|  |  |  |
| --- | --- | --- |
| **Q1D:** | SELECT | \* |
|  | FROM | EMPLOYEE, DEPARTMENT |
|  | WHERE | DNAME='Research' AND DNO=DNUMBER |

**USE OF DISTINCT:**

- SQL does not treat a relation as a set; *duplicate tuples can appear*

- To eliminate duplicate tuples in a query result, the keyword **DISTINCT**

is used

- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

**Q11:** SELECT SALARY FROM EMPLOYEE

**Q11A:** SELECT DISTINCT SALARY FROM EMPLOYEE

**Set Operations, Nesting of Queries, Set Comparisons**

**SET OPERATIONS**

- SQL has directly incorporated some set operations

- There is a union operation (**UNION)**, and in *some versions* of SQL there are set difference (**MINUS)** and intersection (**INTERSECT)** operations

- The resulting relations of these set operations are sets of tuples;

*duplicate tuples are eliminated from the result*

- The set operations apply only to *union compatible relations*; the two relations must have the same attributes and the attributes must appear in the same order

Query 4: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

**Q4:** (SELECT PNAME

FROM PROJECT, DEPARTMENT, EMPLOYEE

|  |  |  |  |
| --- | --- | --- | --- |
| WHERE | DNUM=DNUMBER | AND | MGRSSN=SSN |
| AND  UNION (SELECT | LNAME='Smith')  PNAME |  |  |

FROM PROJECT, WORKS\_ON, EMPLOYEE

WHERE PNUMBER=PNO AND ESSN=SSN AND

**NESTING OF QUERIES:**

- A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*

- Many of the previous queries can be specified in an alternative form using nesting

Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

**Q1:** SELECT FNAME, LNAME, ADDRESS FROM EMPLOYEE

WHERE DNO IN (SELECT DNUMBER FROM DEPARTMENT

WHERE DNAME='Research' )

- The nested query selects the number of the 'Research' department

- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query

- The comparison operator **IN** compares a value v with a set (or multi- set) of values V, and evaluates to **TRUE** if v is one of the elements in V

- In general, we can have several levels of nested queries

- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query*

- In this example, the nested query is *not correlated* with the outer query

**CORRELATED NESTED QUERIES:**

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query*, the two queries are said to be *correlated*

- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*

Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

**Q12:** SELECT E.FNAME, E.LNAME

|  |  |  |
| --- | --- | --- |
| FROM | EMPLOYEE AS E |  |
| WHERE | E.SSN IN (SELECT | ESSN |

FROM DEPENDENT WHERE ESSN=E.SSN AND E.FNAME=DEPENDENT\_NAME)

- In Q12, the nested query has a different result *for each tuple* in the outer query

- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can ***always* be** expressed as a single block query. For example, Q12 may be written as in Q12A

|  |  |  |
| --- | --- | --- |
| **Q12A:** | SELECT | E.FNAME, E.LNAME |
|  | FROM | EMPLOYEE E, DEPENDENT D |
|  | WHERE | E.SSN=D.ESSN AND |

- The original SQL as specified for SYSTEM R also had a **CONTAINS** comparison operator, which is used in conjunction with nested correlated queries

- This operator was dropped from the language, possibly because of the difficulty in implementing it efficiently

- Most implementations of SQL *do not* have this operator

- The CONTAINS operator compares two *sets of values*, and returns

TRUE if one set contains all values in the other set

(Reminiscent of the *division* operation of algebra).

Query 3: Retrieve the name of each employee who works on *all* the projects controlled by department number 5.

|  |  |  |  |
| --- | --- | --- | --- |
| **Q3:** | SELECT | FNAME, LNAME |  |
|  | FROM | EMPLOYEE |
|  | WHERE | ( (SELECT | PNO |
|  |  | FROM WORKS\_ON |  |
|  |  | WHERE | SSN=ESSN) |
|  |  | CONTAINS |  |
|  |  | (SELECT | PNUMBER |
|  |  | FROM PROJECT |  |
|  |  | WHERE | DNUM=5) ) |

**-** In Q3, the second nested query, which is not correlated with the outer query, retrieves the project numbers of all projects controlled by department 5

- The first nested query, which is correlated, retrieves the project numbers on which the employee works, which is different *for each employee tuple* because of the correlation

**The EXISTS function, NULLs, Explicit Sets**

**THE EXISTS FUNCTION:**

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not

- We can formulate Query 12 in an alternative form that uses EXISTS



as Q12B below

Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

**Q12B:** SELECT FNAME, LNAME

|  |  |  |  |
| --- | --- | --- | --- |
| FROM | EMPLOYEE |  | |
| WHERE | EXISTS | (SELECT | \* |
|  |  | FROM | DEPENDENT |
|  |  | WHERE | SSN=ESSN AND |

Query 6: Retrieve the names of employees who have no dependents.

|  |  |  |  |
| --- | --- | --- | --- |
| **Q6:** | SELECT | FNAME, LNAME |  |
|  | FROM | EMPLOYEE |
|  | WHERE | NOT EXISTS (SELECT | \* |

DEPENDENT

FROM

WHERE SSN=ESSN)

- In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist* , the EMPLOYEE tuple is selected

- EXISTS is necessary for the expressive power of SQL

**EXPLICIT SETS:**

**-** It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query

Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

**Q13:** SELECT DISTINCT ESSN FROM WORKS\_ON WHERE PNO IN (1, 2, 3)

**NULLS IN SQL QUERIES:**

- SQL allows queries that check if a value is NULL (missing or undefined or not applicable)

- SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .

Query 14: Retrieve the names of all employees who do not have supervisors.

**Q14:** SELECT FNAME, LNAME FROM EMPLOYEE

WHERE SUPERSSN IS NULL

Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

**Aggregate Functions and Grouping**

**AGGREGATE FUNCTIONS:**

- Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**

Query 15: Find the maximum salary, the minimum salary, and the average salary among all employees.

**Q15:** SELECT MAX (SALARY), MIN (SALARY), AVG (SALARY) FROM EMPLOYEE

- Some SQL implementations *may not allow more than one function* in the SELECT-clause

Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

**Q16:** SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY) FROM EMPLOYEE, DEPARTMENT

WHERE DNO=DNUMBER AND DNAME='Research'

Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

**Q17:** SELECT COUNT (\*) FROM EMPLOYEE

**Q18:** SELECT COUNT (\*)

FROM EMPLOYEE, DEPARTMENT

WHERE DNO=DNUMBER AND DNAME='Research'

**GROUPING**

- In many cases, we want to apply the aggregate functions *to subgroups of tuples in a relation*

- Each subgroup of tuples consists of the set of tuples that have *the same value* for the *grouping attribute(s)*

- The function is applied to each subgroup independently

- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

Query 20: For each department, retrieve the department number, the number of employees in the department, and their average salary.

|  |  |  |
| --- | --- | --- |
| **Q20:** | SELECT | DNO, COUNT (\*), AVG (SALARY) |
|  | FROM | EMPLOYEE |
|  | GROUP BY | DNO |

- In Q20, the EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO

- The COUNT and AVG functions are applied to each such group of tuples separately

- The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples

- A join condition can be used in conjunction with grouping

Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

|  |  |  |
| --- | --- | --- |
| **Q21:** | SELECT | PNUMBER, PNAME, COUNT (\*) |
|  | FROM | PROJECT, WORKS\_ON |
|  | WHERE | PNUMBER=PNO |
|  | GROUP BY | PNUMBER, PNAME |

- In this case, the grouping and functions are applied *after* the joining of the two relations

**THE HAVING-CLAUSE:**

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*

- The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples)

Query 22: For each project *on which more than two employees work* , retrieve the project number, project name, and the number of employees who work on that project.

|  |  |  |
| --- | --- | --- |
| **Q22:** | SELECT | PNUMBER, PNAME, COUNT (\*) |
|  | FROM | PROJECT, WORKS\_ON |
|  | WHERE | PNUMBER=PNO |
|  | GROUP BY | PNUMBER, PNAME |
|  | HAVING | COUNT (\*) > 2 |

**Sub string Comparisons, Arithmetic, ORDER BY**

**SUBSTRING COMPARISON**

- The **LIKE** comparison operator is used to compare partial strings

- Two reserved characters are used: '%' (or '\*' in some implementations) replaces an arbitrary number of characters, and '\_' replaces a single arbitrary character

Query 25: Retrieve all employees whose address is in Houston, Texas.

Here, the value of the ADDRESS attribute must contain the substring

'Houston, TX'.

|  |  |  |
| --- | --- | --- |
| **Q25:** | SELECT | FNAME, LNAME |
|  | FROM | EMPLOYEE |
|  | WHERE | ADDRESS LIKE '%Houston, TX%' |

Query 26: Retrieve all employees who were born during the 1950s.

Here, '5' must be the 8th character of the string (according to our format for date), so the BDATE value is ' 5\_', with each underscore as a place holder for a single arbitrary character.

|  |  |  |
| --- | --- | --- |
| **Q26:** | SELECT | FNAME, LNAME |
|  | FROM | EMPLOYEE |
|  | WHERE | BDATE LIKE '\_ 5\_' |

- The LIKE operator allows us to get around the fact that each value is considered atomic and indivisible; hence, in SQL, character string attribute values are not atomic

**ARITHMETIC OPERATIONS:**

- The standard arithmetic operators '+', '-'. '\*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result

Query 27: Show the effect of giving all employees who work on the

'ProductX' project a 10% raise.

|  |  |  |
| --- | --- | --- |
| **Q27:** | SELECT | FNAME, LNAME, 1.1\*SALARY |
|  | FROM | EMPLOYEE, WORKS\_ON, PROJECT |
|  | WHERE | SSN=ESSN AND PNO=PNUMBER AND |

**ORDER BY**

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)

Query 28: Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

**Q28:** SELECT DNAME, LNAME, FNAME, PNAME

FROM DEPARTMENT, EMPLOYEE, WORKS\_ON, PROJECT

WHERE DNUMBER=DNO AND SSN=ESSN AND ORDER BY DNAME, LNAME

- The default order is in ascending order of values

- We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default