

Databases Illuminated

Chapter 6

Relational Database Management Systems and SQL

History of SQL

- Proposed by E.F.Codd in his 1970 paper
- Used in System R, IBM's research relational database in early 1970s-D.D. Chamberlin et al at IBM Research Center, San Jose, California
- Used in Oracle, released in late 1970s
- Incorporated into IBM's SQL/DS in 1981, and DB2 in 1983
- Also used in Microsoft SQL Server, MySQL, Informix, Sybase, dBase, Paradox, r:Base, FoxPro, and others

Standards

- ANSI and ISO published SQL standards in 1986, called SQL-1
- Minor revision, SQL-89
- Major revision, SQL-2, 1992
- SQL-3, multi-part revision, contains SQL:1999, which included object-oriented (OO) facilities, and user defined datatypes (UDTs)
- Most vendors support standard, but have slight variations of their own

Components of SQL

- Data definition language - DDL
- Data manipulation language - DML
- Authorization language – grant privileges to users

Relational Database Architecture

- Separate external, logical, internal models
- Base tables and indexes form logical level
- Indexes are B+ trees or B trees – maintained by system
- Relational views (external level) are derived from base tables
- Users see views or base tables, or combination
- Internal level - files
- SQL supports dynamic database definition-can modify structures easily
- **See Figure 6.1**

DDL Commands

CREATE TABLE

CREATE INDEX

ALTER TABLE

RENAME TABLE

DROP TABLE

DROP INDEX

Also – CREATE VIEW

CREATE TABLE

```
CREATE TABLE base-table-name (colname datatype  
    [column constraints – NULL/NOT NULL, DEFAULT...,  
    UNIQUE, CHECK..., PRIMARY KEY],  
    [,colname datatype [column constraints ...]]  
    ...  
    [table constraints – PRIMARY KEY..., FOREIGN KEY...,  
    UNIQUE..., CHECK...]  
    [storage specifications]);
```

Identifiers

- No SQL keywords may be used
- Table name must be unique within the database
- For each column, the user must specify a name that is unique within the table

Datatypes

- Each column must have a datatype specified
- Standards include various numeric types, fixed-length and varying-length character strings, bit strings, and user-defined types
- available datatypes vary from DBMS to DBMS
- Oracle types include CHAR(N), VARCHAR2(N), NUMBER(N,D), DATE, and BLOB (binary large object) and others
- DB2 types include SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, REAL, DOUBLE, CHAR(N), VARCHAR(N), LONG VARCHAR, CLOB, GRAPHIC, DBCLOB, BLOB, DATE, TIME, and TIMESTAMP
- SQL Server includes types of NUMERIC, BINARY, CHAR, VARCHAR, DATETIME, MONEY, IMAGE, and others
- Access supports several types of NUMBER, as well as TEXT, MEMO, DATE/TIME, CURRENCY, YES/NO, and others

Creating the Tables for the University Database

```
CREATE TABLE Student
(
  stuld          CHAR(6),
  lastName       CHAR(20) NOT NULL,
  firstName      CHAR(20) NOT NULL,
  major          CHAR(10),
  credits        SMALLINT DEFAULT 0,
  CONSTRAINT Student_stuld_pk PRIMARY KEY (stuld),
  CONSTRAINT Student_credits_cc CHECK (credits >= 0 AND credits < 150));

CREATE TABLE Faculty
(
  facld          CHAR(6),
  name           CHAR(20) NOT NULL,
  department     CHAR(20) NOT NULL,
  rank           CHAR(10),
  CONSTRAINT Faculty_facld_pk PRIMARY KEY (facld));

CREATE TABLE Class
(
  classNumber    CHAR(8),
  facld          CHAR(6) NOT NULL,
  schedule       CHAR(8),
  room           CHAR(6),
  CONSTRAINT Class_classNumber_pk PRIMARY KEY (classNumber),
  CONSTRAINT Class_facld_fk FOREIGN KEY (facld) REFERENCES Faculty (facld)) ;

CREATE TABLE Enroll
(
  stuld          CHAR(6),
  classNumber    CHAR(8),
  grade          CHAR(2),
  CONSTRAINT Enroll_classNumber_stuld_pk PRIMARY KEY (classNumber, stuld),
  CONSTRAINT Enroll_classNumber_fk FOREIGN KEY (classNumber) REFERENCES Class (classNumber),
  CONSTRAINT Enroll_stuld_fk FOREIGN KEY (stuld) REFERENCES Student(stuld));
```

New Datatypes

- SQL:1999 provides new UDTs
- Can define structured UDTs
- Also new DISTINCT types e.g.

```
CREATE DOMAIN creditValues INTEGER  
DEFAULT 0  
CHECK (VALUE >=0 AND VALUE <150);
```

- New type can then be used in defining columns
In Student, can write `credits creditValues,...`
 - Can't compare values of two different DISTINCT types, even if underlying datatype is the same
 - Also can't apply functions to them, but can write our own functions for them
- ...will be discussed more fully in Chapter 7

Constraints

- Can be defined at column level or table level
- Column-level constraints
 - NULL/NOT NULL, UNIQUE, PRIMARY KEY, CHECK and DEFAULT
 - Written immediately after column name, datatype
- Table-level constraints
 - Primary key, foreign keys, uniqueness, checks, and general constraints
 - Note: any primary key can be defined at table level; composite primary keys can only be expressed as table-level constraints
 - Foreign key constraint requires that the referenced table exist already
 - SQL standard supports ON UPDATE and ON DELETE clauses for foreign keys; Options are CASCADE/SET NULL/SET DEFAULT/NO ACTION; Example: ON UPDATE CASCADE
 - Table uniqueness constraint used to specify values in a combination of columns must be unique; good for candidate keys
 - Constraints can be given a name; useful for disabling them at times
 - NOTE: Not all options are supported in all DBMSs

Indexes

- Can create any number of indexes for tables
- Stored in same file as base table
- Facilitate fast retrieval of records with specific values in a column
- Keep track of what values exist for the indexed columns, and which records have those values
- B+ trees or B trees used – see Appendix A for review of concepts
- Overhead – system must maintain index

CREATE INDEX Command

```
CREATE [UNIQUE] INDEX indexname ON  
  basetablename (colname [order] [,colname  
  [order]...) [CLUSTER] ;
```

Ex. CREATE INDEX Student_lastName_firstName_ndx
ON Student (lastName, firstName);

- UNIQUE specification enforces unique values for indexed column or combination of columns
- Except when specified, column need not be unique
- Order is ASC(default) or DESC
- Can have major and minor orders
- CLUSTER specification keeps records with same value for indexed field together (only one per table)
- Oracle automatically indexes primary key columns

ALTER TABLE Command

- To add a new column

ALTER TABLE *basetablename* ADD *columnname datatype*;

Ex. ALTER TABLE Student ADD COLUMN birthdate DATATYPE;

- Cannot specify NOT NULL, since existing records have no value for this field

- To drop a column

ALTER TABLE *basetablename* DROP COLUMN *columnname*;

Ex. ALTER TABLE Student DROP COLUMN major;

- To add a constraint

ALTER TABLE *basetablename* ADD CONSTRAINT *constraint_defn*;

- To drop a constraint

ALTER TABLE *basetablename* DROP CONSTRAINT *constraint_name*;

Other Changes to Tables

- Renaming a table:

RENAME TABLE *old-table-name* TO *new-table-name*;

Ex: RENAME TABLE FACULTY TO TEACHERS;

- Dropping a table:

DROP TABLE *basetablename*;

Ex. DROP TABLE CLASS;

- Dropping an index:

DROP INDEX *indexname*;

Ex. DROP INDEX Student_lastName_fristName_ndx;

SQL DML

- Non-procedural, declarative language
- Can be interactive, can be embedded in host language, or can be stand-alone programming language (SQL/PSMs)
- Basic commands
 - SELECT
 - UPDATE
 - INSERT
 - DELETE

SELECT Statement

SELECT [DISTINCT] *col-name* [AS *newname*], [, *col-name*..]...
FROM *table-name* [*alias*] [, *table-name*]...
[WHERE *predicate*]
[GROUP BY *col-name* [, *col-name*]...[HAVING *predicate*]
or
[ORDER BY *col-name* [, *col-name*]...];

- Powerful command – equivalent to relational algebra's SELECT, PROJECT, JOIN and more...
- Can be applied to one or more tables or views
- Can display one or more columns (renaming if desired)
- Predicate is optional, and may include usual operators and connectives
- Can put results in order by one or more columns
- Can also group together records with the same value for column(s)
- Can also use predefined functions
- See list of examples, Section 6.4

UPDATE Operator

```
UPDATE      tablename  
SET         columnname = expression  
           [columnname = expression]...  
[WHERE predicate];
```

- Used for changing values in existing records
- Can update, zero, one, many, or all records in a table
- For null value, use SET *columnname = NULL*
- can use a sub-query to identify records to be updated

INSERT Operator

INSERT

INTO *tablename* [(*colname* [,*colname*]...)]

VALUES (*constant* [,*constant*]...);

- Used for inserting new records into database, one at a time
- Not necessary to name columns if values are supplied for all columns, in proper order
- To insert null value for a column, specify only the other columns or write *null* as the value
- Can specify values for some columns, in any order, as long as values match order

DELETE Operator

```
DELETE  
FROM      tablename  
WHERE     predicate;
```

- Used for deleting existing records from database
- Can delete zero, one, many, or all records
- Operation may not work if referential integrity would be lost
- Can use a sub-query to target records to be deleted
- If you delete all records from a table, its structure still remains, and you can insert into it later

Relational Views

- Can be subsets of base tables, or subsets of joins, or contain calculated data
- Reasons for views
 - Allow different users to see the data in different forms
 - Provide a simple authorization control device
 - Free users from complicated DML operations
 - If database is restructured, view can keep the user's model constant

Create View

```
CREATE VIEW viewname [(viewcolname  
  [,viewcolname]...)] AS      SELECT      colname  
  [,colname]...  
  FROM      basetablename [,basetablename]...  
  WHERE      condition;
```

- Can create vertical subset of table, choosing only certain columns, with no WHERE, called **value-independent** view
- Can choose only certain rows, using WHERE, called **value-dependent** view
- Can use a join of tables to create view of combination
- Can use functions in SELECT

Using Views

- Can write new SQL statements using view name in FROM line
- Can create a view of a view
- Can sometimes update a view – requires that the primary key be in the view
- When user is allowed to update using view, DB administrator can provide an INSTEAD OF trigger to update base tables instead

```
CREATE TRIGGER InsertStuVw2
INSTEAD OF INSERT ON StudentVw2
FOR EACH ROW
begin
    INSERT
    INTO    Student
    VALUES (:NEW.stuId, :NEW.lastName, :NEW.firstName, :NEW.Credits);
end;
```

Active Databases-Constraints

- DBMS monitors database to prevent illegal states, using constraints and triggers
- Constraints
 - can be specified when table is created, or later
 - IMMEDIATE MODE: constraint checked when each INSERT, DELETE, UPDATE is performed
 - DEFERRED MODE: postpones constraint checking to end of transaction – write SET CONSTRAINT *name* DEFERRED
 - Can use DISABLE CONSTRAINT *name*, and later ENABLE CONSTRAINT *name*

Triggers

- More flexible than constraints
- Must have ECA model:
 - **event**, some change made to the database
 - **condition**, a logical predicate
 - **action**, a procedure done when the event occurs and the condition is true, also called **firing the trigger**
- Can be fired before or after insert, update, delete
- Trigger can access values it needs as :OLD. and :NEW.
 - prefix :OLD refers to values in a tuple deleted or to the values replaced in an update
 - prefix :NEW refers to the values in a tuple just inserted or to the new values in an update.
- Can specify whether trigger fires just once for each triggering statement, or for each row that is changed by the statement

Trigger Syntax

```
CREATE OR REPLACE TRIGGER trigger_name
[BEFORE/AFTER] [INSERT/UPDATE/DELETE] ON
  table_name
[FOR EACH ROW] [WHEN condition]
BEGIN
  trigger body
END;
```

- Can disable triggers using ALTER TRIGGER *name* DISABLE;
- Later write ALTER TRIGGER *name* ENABLE;
- Can drop triggers using DROP TRIGGER *name*;
- See examples in **Figure 6.5**

Ending Transactions

- COMMIT makes permanent changes in the current transaction
- ROLLBACK undoes changes made by the current transaction

SQL Programming

- SQL can be embedded in host languages, such as Java, C++, COBOL, and so on
- Host language can provide control structures, and SQL used for database access
- Executable SQL statements preceded by keyword such as EXEC SQL, end with ;
- Executable SQL statement can appear wherever a host language executable statement can appear
- Pre-compiler for DB compiles SQL separately from program; creates access module
- Host language statements compiled as usual
- Data exchange done using shared variables

Shared Variables

- Declared in SQL declaration section. Ex:
EXEC SQL BEGIN DECLARE SECTION;
char stuNumber[5];
char stuLastName[15];
char stuFirstName[12];
char stuMajor[10];
int stuCredits;
char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
- **SQLSTATE** used for error conditions—'00000' means no error, while '02000' means no tuple found for query
- Value of SQLSTATE should be tested in host language control statements such as WHILE(SQLSTATE='00000') or UNTIL(SQLSTATE='02000')

Single-row Embedded SQL SELECT

```
stuNumber = 'S1001';  
EXEC SQL      SELECT Student.lastName, Student.firstName,  
                  Student.major, Student.credits  
              INTO  :stuLastName, :stuFirstName, :stuMajor, :stuCredits  
              FROM  Student  
              WHERE Student.stuld = :stuNumber;
```

- INTO line lists shared variables declared previously
- In SQL statements, use colon before shared variables to distinguish them from database attributes; they may have the same or different names from attributes
- In the host language, use the shared variables without colon, ex
if(stuCredits>120)...

Insert in Embedded SQL

- Assign values to program shared variables
- Use SQL INSERT, listing shared variables (now preceded by colon) as values

Ex:

```
stuNumber = 'S1050';
stuLastName = 'Lee';
stuFirstName = 'Daphne';
stuMajor = 'English';
stuCredits = 0;
EXEC SQL  INSERT
          INTO Student (stuld, lastName, firstName, major, credits)
          VALUES (:stuNumber, :stuLastName, :stuFirstName, :stuMajor,
                  :stuCredits);
```

Delete in Embedded SQL

- Use program shared variables to identify target tuple(s)
- Use SQL DELETE, identifying tuple(s) in WHERE line using shared variables (preceded by colon)

Ex:

```
stuNumber = 'S1015';  
EXEC SQL      DELETE  
              FROM      Student  
              WHERE stuld = :stuNumber;
```

Update in Embedded SQL

- Use program shared variables to identify target tuple(s)
- Use SQL UPDATE, identifying tuple(s) in WHERE line using shared variables preceded by colon

Ex:

```
stuMajor = 'History';
```

```
EXEC SQL          UPDATE Student
                   SET    CREDITS = CREDITS + 3
                   WHERE major = :stuMajor;
```

Error Handling using WHENEVER

- Can check each SQL transaction individually, or do error-handling for entire program using WHENEVER
- SQLERROR means any exception
- NOT FOUND means SQLSTATE of '02000'

```
EXEC SQL WHENEVER [SQLERROR/NOT  
FOUND][CONTINUE/ GO TO statement];
```

Using Cursors

- Impedance mismatch: SQL SELECT can retrieve multiple rows, while host language requires one row at a time
- Cursor can be created and positioned to provide one tuple at a time to program
- Declaring a cursor:
EXEC SQL DECLARE *cursorname* [INSENSITIVE] [SCROLL] CURSOR FOR *query*
[FOR {READ ONLY | UPDATE OF *attributeNames*}];
- Query is regular SQL query, using attributes names (not the shared variables)
- Opening the cursor executes the SQL query
EXEC SQL OPEN *cursorname*;
- Set up a loop in host language
WHILE (SQLSTATE = '00000')
- To retrieve each row of the results, use FETCH
EXEC SQL FETCH *cursorname* INTO *hostvariables*;
- *hostvariables* are shared variables; preceded by colon
- At end, close the cursor
EXEC SQL CLOSE *cursorname*;

Cursor Example

```
EXEC SQL DECLARE CSCstuCursor CURSOR FOR
    SELECT stuId, lastName, firstName, major, credits
    FROM student
    WHERE major='CSC';
EXEC SQL OPEN CSCstuCursor;
WHILE (SQLSTATE = '00000')
    EXEC SQL FETCH CSCstuCursor INTO
        :stuNumber, :stuLastName, :stuFirstName, :stuMajor, :stuCredits
        //Process each record retrieved
END; //of WHILE
EXEC SQL CLOSE CSCstuCursor;
```

Update and Delete Using a Cursor

- Must declare cursor for update: FOR UPDATE OF *variablename*
- Once cursor is open and active, current of cursor refers to the tuple it is positioned at
- To update:
EXEC SQL UPDATE *tablename*
SET *attributename* = *newvalue*
WHERE CURRENT OF *cursorname*;
- To delete:
EXEC SQL DELETE FROM *tablename*
WHERE CURRENT OF *cursorname*;

Update and Delete Examples

```
EXEC SQL DECLARE stuCreditsCursor CURSOR FOR
  SELECT stuld, credits
  FROM Student
  FOR UPDATE OF credits;
EXEC SQL OPEN stuCreditsCursor;
WHILE (SQLSTATE = '00000')
  EXEC SQL FETCH stuCreditsCursor INTO :stuNumber, ::stuCredits
  EXEC SQL UPDATE Student
    SET credits = credits +3
    WHERE CURRENT OF stuCreditsCursor;
END; //of WHILE loop
```

- For delete, replace update statement by

```
EXEC SQL DELETE FROM Student
  WHERE CURRENT OF stuCreditCursor;
```
- Could target specific tuples in the SQL SELECT statement for declaration of cursor

Dynamic SQL

- Can create a graphical front end that accepts queries dynamically
- At run time, user prompted to enter an SQL command
- Command stored as string
- PREPARE statement used to parse and compile string and assign it to variable
- EXECUTE command executes the code
- Ex

```
char userString[ ]='UPDATE Student SET credits = 36  
WHERE stuld= S1050';
```

```
EXEC SQL PREPARE userCommand FROM :userString;  
EXEC SQL EXECUTE userCommand;
```

API, ODBC and JDBC

- DBMS can provide a library of functions available to host languages using an API
- ODBC/JDBC provide standardized connectivity using a common interface, allows common code to access different databases
- Most vendors provide ODBC or JDBC drivers that conform to the standard
- Requires four components: application, driver manager, driver, and data source (database)

ODBC/JDBC Components

- **Application**, using the standard API
 - initiates the connection with the database
 - submits data requests as SQL statements to the DBMS
 - retrieves the results
 - performs processing
 - terminates the connection
- **Driver manager**
 - loads and unloads drivers at the application's request
 - passes the ODBC or JDBC calls to the selected driver
- **Database driver**
 - links the application to the data source
 - translates the ODBC or JDBC calls to DBMS-specific calls
 - handles data translation needed because of any differences between the DBMS's data language and the ODBC/JDBC standard
 - Controls error handling differences that arise between the data source and the standard.
- **Data source**: database (or other source), DBMS and platform; provides the data

SQL/PSM

- Persistent Stored Modules-used to create internal routines within database space
- Can be saved with database schema and invoked
- Oracle uses PL/SQL, accessed within SQLPlus
- Provides complete programming language facilities-declarations, control, assignment, functions, procedures, temporary relations, etc.

SQL/PSM Procedures and Functions

- Creating procedures
CREATE PROCEDURE *procedure_name* (*parameter_list*)
 declarations of local variables
 procedure code
 - Parameters may be IN, OUT, or IN/OUT
 - Executing procedures-Example:
EXECUTE *procedure_name*(*actual_parameter_list*);
- Creating functions
CREATE FUNCTION *function_name* (*parameter list*) RETURNS
 SQLdatatype
 declarations of local variables
 function code (must include a RETURN statement)
 - Parameters can be IN only, no OUT or IN/OUT
 - A function is invoked by using its name, typically in an assignment statement..Example:
SET newVal = MyFunction(val1, val2);

Declaration, Assignment, Control Statements in SQL/PSM

- DECLARE *identifier datatype*;
- SET *number_of_courses = credits/3*;
- IF (*condition*) THEN *statements*;
 ELSEIF (*condition*) *statements*;
 ...
 ELSEIF (*condition*) *statements*;
 ELSE *statements*;
END IF;
- CASE *selector*
 WHEN *value1* THEN *statements*;
 WHEN *value2* THEN *statements*;
 ...
END CASE;
- LOOP...ENDLOOP, WHILE... DO... END WHILE, REPEAT...UNTIL...END REPEAT, and FOR ...DO... END FOR

System Catalog

- Also called system data dictionary
- Contains metadata
- Automatically updated when new database objects created – stores schema
- Oracle data dictionary provides three views: USER, ALL, and DBA.
- View invoked by using the appropriate term as a prefix for the object(s) named in the FROM clause in a query
 - **USER view** provides a user with information about all the objects created by that user
 - **ALL view** provides information about objects user has permission to access in addition to the one the user has created.
 - **DBA view** provides information about all database objects; available to the database administrator

Using Oracle Data Dictionary- Examples

```
DESCRIBE STUDENT;  
DESCRIBE USER_CONSTRAINTS;
```

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME  
FROM USER_CONSTRAINTS;
```

```
SELECT TABLE_NAME  
FROM USER_TABLES;
```

```
SELECT VIEW_NAME  
FROM USER_VIEWS;
```

```
SELECT TRIGGER_NAME, TRIGGER_EVENT, TRIGGER_TYPE  
FROM USER_TRIGGERS;
```

```
SELECT *  
FROM USER_TAB_COLUMNS;
```

```
SELECT COLUMN_NAME, DATA_TYPE  
FROM USER_TAB_COLUMNS  
WHERE TABLE_NAME = 'STUDENT';
```

IBM DB2 UDB Catalog

- system catalog in a schema called SYSIBM, usually with restricted access
- Two views, SYSCAT and SYSSTAT, are available for users
- **SYSCAT schema:**
 - TABLES(TABSHEMA, TABNAME, DEFINER, TYPE, STATUS, COLCOUNT, KEYCOLUMNS, CHECKCOUNT,...)
 - COLUMNS(TABSHEMA, TABNAME, COLNAME, COLNO, TYPENAME, LENGTH, DEFAULT, NULLS, ...)
 - INDEXES(INDSCHEMA, INDNAME, DEFINER, TABSCHEMA, TABNAME, COLNAMES, UNIQUERULE, COLCOUNT, ...)
 - TRIGGERS(TRIGSCHEMA, TRIGNAME, DEFINER, TABSCHEMA, TABNAME, TRIGTIME, TRIGEVENT, ...)
 - VIEWS(VIEWSCHEMA VIEWNAME, DEFINER, TEXT...)
- Can query this database using usual SQL SELECT statements

```
SELECT TABLESCHEMA, TABLENAME
FROM TABLES
WHERE DEFINER = 'JONES';

SELECT *
FROM COLUMNS
WHERE TABNAME = 'STUDENT'
GROUP BY COLNAME;
```