



高级SQL

Advanced SQL

李文根/Wengen Li

Email: lwengen@tongji.edu.cn

先进数据与机器智能系统实验室

Advanced Data and Machine Intelligence Systems (ADMIS) Lab

<https://admis-tongji.github.io>

同济大学 计算机科学与技术学院

2025年03月

- **Part 0: Overview**
 - Ch1: Introduction
- **Part 1 Relational Languages**
 - Ch2: Relational model
 - Ch3: Introduction to SQL
 - Ch4: Intermediate SQL
 - **Ch5: Advanced SQL**
- **Part 2 Database Design**
 - Ch6: Database design via E-R model
 - Ch7: Relational database design
- **Part 3 Application Design & Development**
 - Ch8: Complex data types
 - Ch9: Application development
- **Part 4 Big Data Analytics**
 - Ch10: Big data
 - Ch11: Data analytics
- **Part 5 Storage Management & Indexing**
 - Ch12: Physical storage systems
 - Ch13: Data storage structures
 - Ch14: Indexing
- **Part 6 Query Processing & Optimization**
 - Ch15: Query processing
 - Ch16: Query optimization
- **Part 7 Transaction Management**
 - Ch17: Transactions
 - Ch18: Concurrency control
 - Ch19: Recovery system
- **Part 8 Parallel & Distributed Database**
 - Ch20: Database system architecture
 - Ch21-23: Parallel & distributed storage, query processing & transaction processing
- **Advanced topics**
 - DB Platform: **OceanBase**, MongoDB, Neo4J
 - RAG, Multimodal retrieval, ...

- **编程访问SQL**
- 函数和过程
- 触发器
- 递归查询
- 高级聚集特性

- API (application-program interface) for a program to interact with a database server
- **Application makes calls to**
 - Connect with the database server
 - Send SQL commands to the database server
 - Fetch tuples of result one-by-one into program variables
- **Various tools:**
 - Dynamic SQL
 - **ODBC** (Open Database Connectivity) works with C, C++, C#, and Visual Basic. Other API's such as ADO.NET sit on top of ODBC
 - **JDBC** (Java Database Connectivity) works with Java
 - Embedded SQL

▶ JDBC (Java Database Connectivity)



- **JDBC**
 - a Java API for communicating with database systems supporting SQL
 - support a variety of features for querying and updating data, and for retrieving query results
 - support metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes
- **Model for communicating with the database:**
 - Open a connection
 - Create a “Statement” object
 - Execute queries using the Statement object to send queries and fetch results
 - Exception mechanism to handle errors

```
public static void JDBCexample(String dbid, String userid, String passwd){
    try {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@aura.bell-
labs.com:2000:bankdb", userid, passwd);
        Statement stmt = conn.createStatement();
        ... Do Actual Work ....
        stmt.close();
        conn.close();
    }
    catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle);
    }
}
```

- **Update to database**

```
try { stmt.executeUpdate( "insert into account values  
                        ('A-9732', 'Perryridge', 1200)");  
}  
catch (SQLException sqle) {  
    System.out.println("Could not insert tuple. " + sqle);  
}
```

- **Execute query and fetch and print results**

```
ResultSet rset = stmt.executeQuery( "select branch_name, avg(balance)  
    from account group by branch_name");  
while (rset.next()) {  
    System.out.println(  
        rset.getString("branch_name") + " " + rset.getFloat(2));  
}
```

- Getting result fields:
rs.getString("branch_name") and *rs.getString(1)* are equivalent if *branch_name* is the first argument of select result.
- Dealing with Null values
if (rs.isNull())
Systems.out.println("Got null value");

- **Open DataBase Connectivity(ODBC) standard**
 - standard for application program to communicate with a database server
 - application program interface (API) to
 - open a connection with a database
 - send queries and updates
 - get back results
- Applications such as GUI, statistical analysis, and spreadsheets can use ODBC

▶ ODBC (Cont.)



- Each database system supporting ODBC provides a "driver" library that must be linked with the client program
- When client program makes an ODBC API call, the code in the library communicates with the server to carry out the requested action, and fetch results
- ODBC program first allocates an SQL environment, then a database connection handle

▶ ODBC (Cont.)



- Opens database connection using `SQLConnect()`. Parameters for `SQLConnect`:
 - connection handle
 - the server to which to connect
 - the user identifier
 - password
- Must also specify types of arguments:
 - Constant (常数) `SQL_NTS` denotes that previous argument is a null-terminated string

```
int ODBCexample(){
    RETCODE error;
    HENV  env; /* environment */
    HDBC  conn; /* database connection */
    SQLAllocEnv(&env);
    SQLAllocConnect(env, &conn);
    SQLConnect(conn, "db.yale.edu", SQL_NTS, "avi", SQL_NTS, "avipasswd", SQL_NTS);
    { .... Do actual work ... }
    SQLDisconnect(conn);
    SQLFreeConnect(conn);
    SQLFreeEnv(env);
}
```

▶ ODBC Code (Cont.)



- Program sends SQL commands to the database by using `SQLExecDirect`
- Result tuples are fetched using `SQLFetch()`
- `SQLBindCol()` binds C language variables to attributes of the query result
 - When a tuple is fetched, its attribute values are automatically stored in corresponding C variables

- **Arguments to SQLBindCol()**
 - ODBC stmt variable, attribute position in query result
 - The type conversion from SQL to C
 - The address of the variable
 - For variable-length types like character arrays
 - The maximum length of the variable
 - Location to store actual length when a tuple is fetched
 - Note: A negative value returned for the length field indicates null value
- Good programming requires checking results of every function call for errors; we have omitted most checks for brevity

- **Main body of program**

```
char branchname[80];
float balance;
int lenOut1, lenOut2;
HSTMT stmt;
SQLAllocStmt(conn, &stmt);
char * sqlquery = "select branch_name, sum (balance)
                  from account group by branch_name";
error = SQLExecDirect(stmt, sqlquery, SQL_NTS);
if (error == SQL_SUCCESS) {
    SQLBindCol(stmt, 1, SQL_C_CHAR, branchname, 80, &lenOut1);
    SQLBindCol(stmt, 2, SQL_C_FLOAT, &balance, 0, &lenOut2);
    while (SQLFetch(stmt) >= SQL_SUCCESS) {
        printf (" %s %g\n", branchname, balance);
    }
}
SQLFreeStmt(stmt, SQL_DROP);
```

- **Prepared Statement**
 - SQL statement prepared: compiled at the database
 - Can have placeholders: E.g. insert into account values(?,?,?)
 - Repeatedly executed with actual values for the placeholders
- By default, each SQL statement is treated as a separate transaction that is committed automatically
 - Can turn off automatic commit on a connection
 - `SQLSetConnectOption(conn, SQL_AUTOCOMMIT, 0)`
 - transactions must then be committed or rolled back explicitly by
 - `SQLTransact(conn, SQL_COMMIT)` or
 - `SQLTransact(conn, SQL_ROLLBACK)`

- The SQL standard defines embeddings of SQL in a variety of programming languages such as C, and Java
- A language to which SQL queries are embedded is referred to as a **host language** (宿主语言), and the SQL structures permitted in the host language comprise embedded SQL
- EXEC SQL statement is used to identify embedded SQL request to the preprocessor

EXEC SQL <embedded SQL statement > END_EXEC

Note: this varies by language (for example, the Java embedding uses # SQL { };)

▶ Example Query



- Find the names and cities of customers with more than the variable amount dollars in some account
- Specify the query in SQL and declare a cursor for it

EXEC SQL

```
declare c cursor for  
select depositor.customer_name, customer_city  
from depositor, customer, account  
where depositor.customer_name = customer.customer_name  
and depositor account_number = account.account_number  
and account.balance > :amount
```

END_EXEC

► Embedded SQL (Cont.)



- The open statement causes the query to be evaluated

```
EXEC SQL open c END_EXEC
```

- The fetch statement causes the values of one tuple in the query result to be placed on host language variables.

```
EXEC SQL fetch c into :cn, :cc END_EXEC
```

Repeated calls to fetch get successive tuples in the query result

- The close statement causes the database system to delete the temporary relation that holds the result of the query

```
EXEC SQL close c END_EXEC
```

- Note: above details vary with language. For example, the Java embedding defines Java iterators to step through result tuples.

► Updates Through Cursors



- Can update tuples fetched by cursor by declaring that the cursor is for update

```
declare c cursor for  
select *  
from account  
where branch_name = 'Perryridge'  
for update
```

- To update tuple at the current location of cursor c

```
update account  
set balance = balance + 100  
where current of c
```

► Dynamic SQL



- Allows programs to construct and submit SQL queries at run time
- Example of the use of dynamic SQL within a C program.

```
char * sqlprog = "update account  
  set balance = balance * 1.05  
  where account_number = ?"
```

```
EXEC SQL prepare dynprog from :sqlprog;  
char account [10] = "A-101";  
EXEC SQL execute dynprog using :account;
```
- The dynamic SQL program contains a ?, which is a placeholder for a value that is provided when the SQL program is executed

- 编程访问SQL
- **函数和过程**
- 触发器
- 递归查询
- 高级聚集特性

- **SQL provides a module language**
 - Permits definition of procedures in SQL, with if-then-else statements, for and while loops, etc.
- **Stored procedures**
 - can store procedures in the database
 - then execute them using the call statement
 - permit external applications to operate on the database without knowing about internal details

- SQL:1999 supports functions and procedures
 - Functions/procedures can be written in SQL itself, or in an external programming language
 - Functions are particularly useful with specialized data types such as images and geometric objects
 - Example: functions to check if polygons overlap, or to compare images for similarity
 - Some database systems support table-valued functions, which can return a relation as a result
- SQL:1999 also supports a rich set of imperative constructs, including
 - loops, if-then-else, assignment
- Many databases have proprietary procedural extensions to SQL that differ from SQL:1999

- Define a function that, given the name of a customer, returns the count of the number of accounts owned by the customer.

```
create function account_count (customer_name varchar(20)) returns integer  
begin  
    declare a_count integer;  
    select count (*) into a_count  
    from depositor  
    where depositor.customer_name = customer_name  
    return a_count;  
end
```

- Find the name and address of each customer that has more than one account

```
select customer_name, customer_street, customer_city  
from customer  
where account_count (customer_name ) > 1
```

► Table Functions (表函数)



- SQL:2003 added functions that return a relation as a result

- Example: return all accounts owned by a given customer

create function *accounts_of* (*customer_name* char(20)

returns table (*account_number* char(10),

branch_name char(15)

balance numeric(12,2))

return table

(*select* *account_number*, *branch_name*, *balance*

from *account* *A*

where exists (

select *

from *depositor* *D*

where *D.customer_name* = *accounts_of.customer_name*

and *D.account_number* = *A.account_number*))

- Usage: *select* * *from table* (*accounts_of* ('Smith'))

- The `author_count` function could instead be written as procedure:

```
create procedure account_count_proc (in customer_name varchar(20), out a_count integer)
begin
  select count(*) into a_count
  from depositor
  where depositor.customer_name =account_count_proc.customer_name
end
```

- Procedures can be invoked either from an SQL procedure or from embedded SQL, using the call statement.

```
declare a_count integer;
call account_count_proc( 'Smith', a_count);
```

Procedures and functions can be invoked also from dynamic SQL

- SQL:1999 allows more than one function/procedure of the same name (called name overloading), as long as the numbers of arguments differ, or at least the types of the arguments differ

► Procedural Constructs



- Compound statement: begin ... end
 - May contain multiple SQL statements between begin and end
 - Local variables can be declared within a compound statements
- While and repeat statements:

declare n integer default 0;

while n < 10 do

set n = n + 1

end while

repeat

set n = n - 1

until n = 0

end repeat

► Procedural Constructs (Cont.)



- **For loop**

- Permits iteration over all results of a query
- E.g., find total of all balances at the Perryridge branch

```
declare n integer default 0;  
for r as  
    select balance from account  
    where branch_name = 'Perryridge'  
do  
    set n = n + r.balance  
end for
```

- Conditional statements (if-then-else)
 - E.g. To find sum of balances for each of three categories of accounts (with balance <1000, >=1000 and <5000, >= 5000)

```
if r.balance < 1000  
  then set l = l + r.balance  
elseif r.balance < 5000  
  then set m = m + r.balance  
else set h = h + r.balance  
end if
```

- Signaling of exception conditions, and declaring handlers for exceptions

```
declare out_of_stock condition  
declare exit handler for out_of_stock  
begin  
...  
.. signal out-of-stock  
end
```

- The handler here is exit -- causes enclosing begin...end to be exited
- Other actions possible on exception

▶ External Language Functions/Procedures



- SQL:1999 permits the use of functions and procedures written in other languages such as C or C++
- Declaring external language procedures and functions

```
create procedure account_count_proc(in customer_name varchar(20), out count integer)  
language C  
external name ' /usr/avi/bin/account_count_proc '
```

```
create function account_count(customer_name varchar(20))  
returns integer  
language C  
external name ' /usr/avi/bin/author_count '
```

► External Language Routines (Cont.)



- Benefits of external language functions/procedures:
 - more efficient for many operations, and more expressive power
- Drawbacks
 - Code to implement function may need to be loaded into database system and executed in the database system's address space
 - risk of accidental corruption of database structures
 - security risk, allowing users access to unauthorized data
 - Direct execution in the database system's space is used when efficiency is more important than security

- To deal with security problems
 - Use sandbox techniques
 - that is use a safe language like Java, which cannot be used to access/damage other parts of the database code
 - Or, run external language functions/procedures in a separate process, with no access to the database process' memory
 - Parameters and results communicated via inter-process communication
- Both have performance overheads
- Many database systems support both above approaches as well as direct executing in database system address space

- 编程访问SQL
- 函数和过程
- **触发器**
- 递归查询
- 高级聚集特性

► Triggers (触发器)



- A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database
- To design a trigger mechanism, we should:
 - specify the **conditions** under which the trigger is to be executed
 - specify the **actions** to be taken when the trigger executes
- The above model of triggers is referred to as the **event-condition-action** (ECA) model for triggers

▶ Trigger Example



- Suppose that instead of allowing negative account balances, the bank deals with overdrafts (透支) by
 - setting the account balance to zero
 - creating a loan in the amount of the overdraft
 - giving this loan a loan number identical to the account number of the overdrawn account
- The **condition** for executing the trigger is an update (event) to the account relation that results in a negative balance value

▶ Trigger Example in SQL:1999



```
create trigger overdraft_trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
  insert into borrower
    (select customer_name, account_number
     from depositor
     where nrow.account_number = depositor.account_number);
  insert into loan values
    (nrow.account_number, nrow.branch_name, -nrow.balance);
  update account set balance = 0
  where account.account_number = nrow.account_number
end
```

▶ Triggering Events and Actions in SQL



- Triggering event can be insert, delete or update
- Triggers on update can be restricted to specific attributes
 - E.g., after update of balance on account
- Values of attributes before and after an update can be referenced
 - referencing old row as: for deletes and updates
 - referencing new row as: for inserts and updates

▶ Triggering Events and Actions in SQL



- Triggers can be activated before an event, which can serve as extra constraints

create trigger setnull_trigger before update on r

referencing new row as nrow

for each row

when nrow.phone_number = ''

set nrow.phone_number = null

▶ Statement Level Triggers



- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
 - Use for each statement instead of for each row
 - Use referencing old table or referencing new table to refer to temporary tables (called transition tables) containing the affected rows
 - Can be more efficient when dealing with SQL statements that update a large number of rows

► External World Actions



- We sometimes require external world actions to be triggered on a database update
 - E.g. re-ordering an item whose quantity in a warehouse has become small, or turning on an alarm light,
- Triggers cannot be used to directly implement external world actions, BUT
 - Triggers can be used to record actions-to-be-taken in a separate table
 - Have an external process that repeatedly scans the table, carries out external world actions and deletes action from table

► External World Actions



- E.g., suppose a warehouse has the following tables
 - inventory(item, level): How much of each item is in the warehouse
 - minlevel(item, level) : The minimum desired level
 - reorder(item, amount): What quantity should we re-order
 - orders(item, amount) : Orders to be placed

▶ External World Actions (Cont.)



```
create trigger reorder_trigger after update of amount on inventory
referencing old row as orow, new row as nrow
for each row
  when nrow.level <= (select level
                      from minlevel
                      where minlevel.item = orow.item)
  and orow.level > (select level
                   from minlevel
                   where minlevel.item = orow.item)
begin
  insert into orders
    (select item, amount
     from reorder
     where reorder.item = orow.item)
end
```

▶ When Not to Use Triggers



- Triggers were used earlier for tasks such as
 - maintaining **summary data** (e.g., total salary of each department)
 - **Replicating databases** by recording changes to special relations and having a separate process that applies the changes over to a replica
- There are better ways of doing these now:
 - Databases today provide built in **materialized view** facilities to maintain summary data
 - Databases provide built-in support for replication
- Encapsulation facilities can be used instead of triggers in many cases
 - Define methods to update fields
 - Carry out actions as part of the update methods instead of through a trigger

- 编程访问SQL
- 函数和过程
- 触发器
- **递归查询**
- 高级聚集特性

- SQL:1999 permits recursive view definition
 - E.g., find all employee-manager pairs, where the employee reports to the manager directly or indirectly (that is manager's manager, manager's manager's manager, etc.)

```
with recursive empl (employee_name, manager_name ) as (  
    select employee_name, manager_name  
    from manager /*a base query */  
union  
    select manager.employee_name, empl.manager_name  
    from manager, empl /*a recursive query*/  
    where manager.manager_name = empl.employee_name)  
select *  
from empl
```

Note: This example view **empl** is called the transitive closure (传递闭包) of the manager relation

► The Power of Recursion



- Recursive views make it possible to write queries, such as transitive closure queries, that cannot be written without recursion or iteration.
 - Intuition: Without recursion, a non-recursive non-iterative program can perform only a fixed number of joins of manager with itself
 - This can give only a fixed number of levels of managers
 - Given a program we can construct a database with a greater number of levels of managers on which the program will not work

- **Computing transitive closure**
 - The next slide shows a manager relation
 - Each step of the iterative process constructs an extended version of empl from its recursive definition.
 - The final result is called the *fixed point* of the recursive view definition.
- Recursive views are required to be monotonic. That is, if we add tuples to manger the view contains all of the tuples it contained before, plus possibly more

▶ Example of Fixed-Point Computation



<i>employee_name</i>	<i>manager_name</i>
Alon	Barinsky
Barinsky	Estovar
Corbin	Duarte
Duarte	Jones
Estovar	Jones
Jones	Klinger
Rensal	Klinger

<i>Iteration number</i>	<i>Tuples in empl</i>
0	
1	(Duarte), (Estovar)
2	(Duarte), (Estovar), (Barinsky), (Corbin)
3	(Duarte), (Estovar), (Barinsky), (Corbin), (Alon)
4	(Duarte), (Estovar), (Barinsky), (Corbin), (Alon)

- 编程访问SQL
- 函数和过程
- 触发器
- 递归查询
- **高级聚集特性**

- Create a table with the same schema as an existing table:
create table temp_account like account
- SQL:2003 allows subqueries to occur anywhere a value is required provided the subquery returns only one value. This applies to updates as well
- SQL:2003 allows subqueries in the from clause to access attributes of other relations in the from clause using the lateral construct:

```
select C.customer_name, num_accounts  
from customer C,  
lateral (select count(*)  
from account A  
where A.customer_name = C.customer_name )  
as this_customer (num_accounts )
```

▶ Advanced SQL Features (Cont.)



- Merge construct allows batch processing of updates
 - E.g., relation `funds_received` (`account_number`, `amount`) has a batch of deposits to be added to the proper account in the `account` relation

merge into account as A

*using (select **

from funds_received) as F

on (A.account_number = F.account_number)

when matched then

update set balance = balance + F.amount