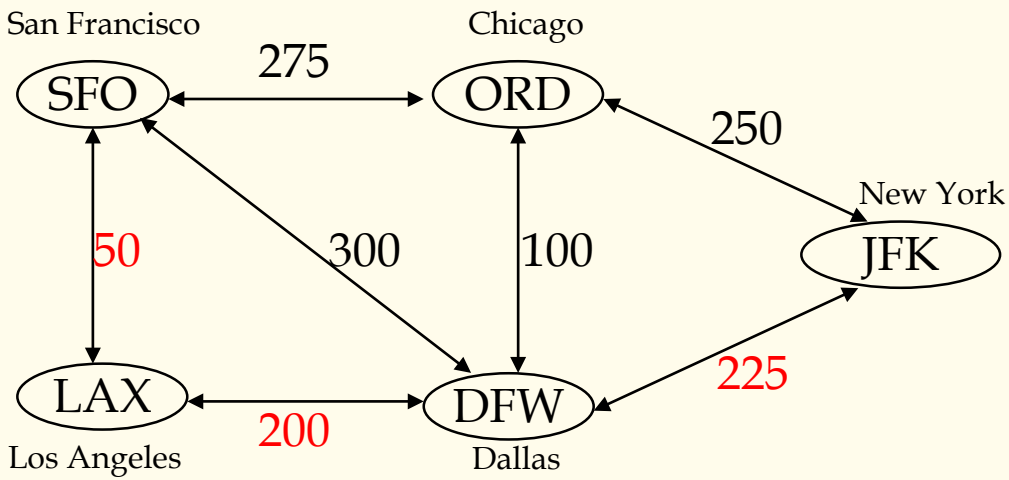
3. User Interfaces and SQL Language (4/4)

Recursive Search

- Typical airline route searching problem
- *Find the lowest total cost route from SFO to JFK*



Flights

FlightNo	Origin	Destination	Cost
HY 120	DFW	JFK	225
HY 130	DFW	LAX	200
HY 140	DFW	ORD	100
HY 150	DFW	SFO	300
HY 210	JFK	DFW	225
HY 240	JFK	ORD	250
HY 310	LAX	DFW	200
HY 350	LAX	SFO	50
HY 410	ORD	DFW	100
HY 420	ORD	JFK	250
HY 450	ORD	SFO	275
HY 510	SFO	DFW	300
HY 530	SFO	LAX	50
HY 540	SFO	ORD	275

Recursive Search

```
WITH trips (destination, route, nsegs, totalcost) AS
     ((SELECT destination, CAST(destination AS varchar(20)), 1, cost
       FROM flights                                          --- initial query
       WHERE origin='SFO')
     UNION ALL
      (SELECT f.destination,                                 --- recursive query
                  CAST(t.route||','||f.destination AS varchar(20)),
                  t.nsegs+1, t.totalcost+f.cost
       FROM trips t, flights f
       WHERE t.destination=f.origin
                  AND f.destination<>'SFO'                   --- stopping rule 1
                  AND f.origin<>'JFK'                        --- stopping rule 2
                  AND t.nsegs<=3))                           --- stopping rule 3
SELECT route,  totalcost                                     --- final query
FROM trips
WHERE destination='JFK' AND totalcost=                       --- lowest cost rule
                                  (SELECT min(totalcost)
                                   FROM trips
                                   WHERE destination='JFK') ;
```

Result

Trips

Destination	Route	Nsegs	Totalcost
DFW	DFW	1	300
ORD	ORD	1	275
LAX	LAX	1	50
JFK	DFW, JFK	2	525
LAX	DFW, LAX	2	500
ORD	DFW, ORD	2	400
DFW	LAX, DFW	2	250
DFW	ORD, DFW	2	375
JFK	ORD, JFK	2	525
DFW	DFW, LAX, DFW	3	700
DFW	DFW, ORD, DFW	3	500
JFK	DFW, ORD, JFK	3	650
LAX	LAX, DFW, LAX	3	450
JFK	LAX, DFW, JFK	3	475
ORD	LAX, DFW, ORD	3	350
LAX	ORD, DFW, LAX	3	575
JFK	ORD, DFW, JFK	3	600
ORD	ORD, DFW, ORD	3	475

Final result

route	totalcost
LAX, DFW, JFK	475

Recursive Search

- *Only change the final query slightly, the least transfer time routes can be found :*

… …

```
SELECT route,  totalcost                          --- final query
FROM trips
WHERE destination='JFK' AND nsegs=                --- least stop rule
                              (SELECT min(nsegs)
                               FROM trips
                               WHERE destination='JFK') ;
```

Final result

route	totalcost
DFW, JFK	525
ORD, JFK	525

Data Manipulation Language

- Insert
  - Insert a tuple into a table
    - *INSERT INTO EMPLOYEES VALUES ('Smith', 'John', '1980-06-10', 'Los Angles', 16, 45000);*
- Delete
  - Delete tuples fulfill qualifications
    - *DELETE FROM Person WHERE LastName = 'Rasmussen' ;*
- Update
  - Update the attributes' value of tuples fulfill qualifications
    - *UPDATE Person SET Address = 'Zhongshan 23', City = 'Nanjing' WHERE LastName = 'Wilson';*

View in SQL

- General view
  - Virtual tables derived from base tables
  - Logical data independence
  - Security of data
  - Update problems of view
- Temporary view and recursive query
  - WITH
  - RECURSIVE

Update problems of view

- CREATE VIEW YoungSailor AS

  SELECT sid, sname, rating

  FROM Sailors

  WHERE age<26;

- CREATE VIEW Ratingavg AS

  SELECT rating, AVG(age)

  FROM Sailors

  GROUP BY rating;

Embedded SQL

- In order to access database in programs, and take further process to the query results, need to combine SQL and programming language (such as C / C++, etc.)
- Problems should be solved:
  - How to accept SQL statements in programming language
  - How to exchange data and messages between programming language and DBMS
  - The query result of DBMS is a set, how to transfer it to the variables in programming language
  - The data type of DBMS and programming language may not the same exactly.

General Solutions

- Embedded SQL
  - The most basic method. Through pre-compiling, transfer the embedded SQL statements to inner library functions call to access database.
- Programming APIs
  - Offer a set of library functions or DLLs to programmer directly, linking with application program while compiling.
- Class Library
  - Supported after emerging of OOP. Envelope the library functions to access database as a set of class, offering easier way to treat database in programming language.

Usage of Embedded SQL (in C)

- SQL statements can be used in C program directly:
  - Begin with *EXEC SQL*, end with '*;*'
  - Through *host variables* to transfer information between C and SQL. Host variables should be defined begin with *EXEC SQL*.
  - In SQL statements, should add ':' before host variables to distinguish with SQL's own variable or attributes' name.
  - In host language (such as C), host variables are used as general variables.
  - Can't define host variables as Array or Structure.
  - A special host variable, SQLCA (SQL Communication Area)

    EXEC SQL INCLUDE SQLCA
  - Use SQLCA.SQLCODE to justify the state of result.
  - Use *indicator* (short int) to treat *NULL* in host language.

Example of *host variables* defining

```
EXEC SQL BEGIN DECLARE SECTION;
    char SNO[7];
    char GIVENSNO[7];
    char CNO[6];
    char GIVENCNO[6];
    float GRADE;
    short GRADEI;          /*indicator of GRADE*/
EXEC SQL END DECLARE SECTION;
```

Executable Statements

- CONNECT
  - EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;
- Execute DDL or DML Statements
  - EXEC SQL INSERT INTO SC(SNO,CNO,GRADE)
    VALUES(:SNO, :CNO, :GRADE);
- Execute Query Statements
  - EXEC SQL SELECT GRADE
    INTO :GRADE :GRADEI
    FROM SC
    WHERE SNO=:GIVENSNO AND
    CNO=:GIVENCNO;
- Because {SNO,CNO} is the key of SC, the result of this query has only one tuple. How to treat result if it has a set of tuples?

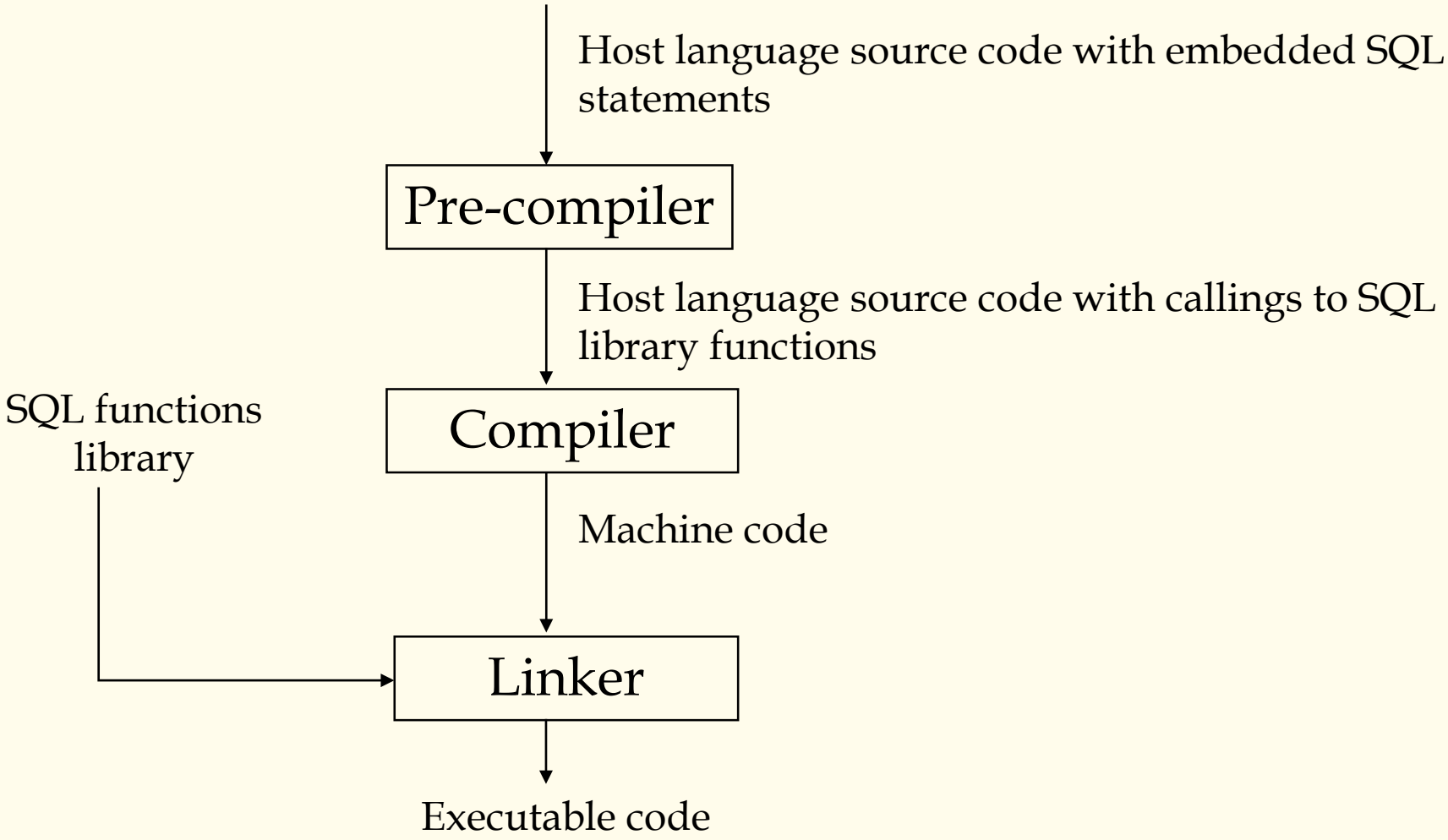
Cursor

1. Define a cursor
   - EXEC SQL DECLARE <*cursor name*> CURSOR FOR
     SELECT …
     FROM …
     WHERE …
2. EXEC SQL OPEN <*cursor name*>
   - Some like open a file
3. Fetch data from cursor
   - EXEC SQL FETCH <*cursor name*>
     INTO :hostvar1, :hostvar2, …;
4. SQLCA.SQLCODE will return 100 when arriving the end of cursor
5. CLOSE CURSOR <*cursor name*>

Example of Query with Cursor

```
        :
EXEC SQL DECLARE C1 CURSOR FOR
        SELECT SNO, GRADE
        FROM SC
        WHERE CNO = :GIVENCNO;
EXEC SQL OPEN C1;
if (SQLCA.SQLCODE<0) exit(1);          /* There is error in query*/
while (1) {
    EXEC SQL FETCH C1 INTO :SNO, :GRADE :GRADEI
    if (SQLCA.SQLCODE==100) break;
    /* treat data fetched from cursor, omitted*/
        :
}
EXEC SQL CLOSE C1;
        :
```

Conceptual Evaluation



Dynamic SQL

- In above embedded SQL, the SQL statements must be written before compiling. But in some applications, the SQL statement can't decided in ahead, they need to be built dynamically while the program running.
- Dynamic SQL is supported in SQL standard and most RDBMS pruducts
  - Dynamic SQL executed directly
  - Dynamic SQL with dynamic parameters
  - Dynamic SQL for query

Dynamic SQL executed directly

- Only used in the execution of non query SQL statements

```
        :
EXEC SQL BEGIN DECLARE SECTION;
char sqlstring[200];
EXEC SQL END DECLARE SECTION;
char cond[150];
strcpy( sqlstring, "DELETE FROM STUDENT WHERE ");
printf(" Enter search condition :");
scanf("%s", cond);
strcat( sqlstring, cond);
EXEC SQL EXECUTE IMMEDIATE :sqlstring;
        :
```

Dynamic SQL with dynamic parameters

- Only used in the execution of non query SQL statements. Use ***place holder*** to realize dynamic parameter in SQL statement. Some like the macro processing method in C.

```
        :
EXEC SQL BEGIN DECLARE SECTION;
char sqlstring[200];
int birth_year;
EXEC SQL END DECLARE SECTION;
strcpy( sqlstring, "DELETE FROM STUDENT WHERE
                        YEAR(BDATE) <= :y; ");
printf(" Enter birth year for delete :");
scanf("%d", &birth_year);
EXEC SQL PREPARE purge FROM :sqlstring;
EXEC SQL EXECUTE purge USING :birth_year;
        :
```

Dynamic SQL for query

- Used to form query statement dynamically

:

```
EXEC SQL BEGIN DECLARE SECTION;
char sqlstring[200];
char SNO[7];
float GRADE;
short GRADEI;
char GIVENCNO[6];
EXEC SQL END DECLARE SECTION;
char orderby[150];
strcpy( sqlstring, "SELECT SNO,GRADE FROM SC WHERE CNO= :c ");
printf(" Enter the ORDER BY clause :");
scanf("%s", orderby);
strcat( sqlstring, orderby);
printf(" Enter the course number :");
scanf("%s", GIVENCNO);
EXEC SQL PREPARE query FROM :sqlstring;
EXEC SQL DECLARE grade_cursor CURSOR FOR query;
EXEC SQL OPEN grade_cursor USING :GIVENCNO;
```

Dynamic SQL for query (Cont.)

```
if (SQLCA.SQLCODE<0) exit(1);                    /* There is error in query*/
while (1) {
    EXEC SQL FETCH grade_cursor INTO :SNO, :GRADE :GRADEI
    if (SQLCA.SQLCODE==100)      break;
    /* treat data fetched from cursor, omitted*/
                :
}
EXEC SQL CLOSE grade_cursor;
                :
```

Stored Procedure

- Used to improve performance and facilitate users. With it, user can take frequently used database access program as a procedure, and store it in the database after compiling, then call it directly while need.
  - Make user convenient. User can call them directly and don't need code again. They are reusable.
  - Improve performance. The stored procedures have been compiled, so they don't need parsing and query optimization again while being used.
  - Expand function of DBMS. (can write script)

Example of a Stored Procedure

```
EXEC SQL
    CREATE PROCEDURE drop_student
          (IN student_no CHAR(7),
           OUT message CHAR(30))
    BEGIN ATOMIC
          DELETE FROM STUDENT
                    WHERE SNO=student_no;
          DELETE FROM SC
                    WHERE SNO=student_no;
          SET message=student_no || 'droped';
    END;
EXEC SQL

          :
CALL drop_student(…);          /* call this stored procedure later*/
          :
```