Friday - Practical Tables

CS530 Database Architecture Models and Design

Prof. Ian HORROCKS
Dr. Robert Stevens
In this Section...

- Topics Covered
  - PL-SQL Query and View
  - Language Connectivity
- Examples Classes
PL/SQL

- PL/SQL is Oracle's procedural language extension to SQL.
- PL/SQL combines SQL with the procedural functionality of a structured programming language, such as IF ... THEN, WHILE, and LOOP.
- The PL/SQL engine used to define, compile, and execute PL/SQL program units.
- A component of many Oracle products, including Oracle Server.
Procedures and Functions

- A set of SQL and PL/SQL statements grouped together as a unit (*block*) to solve a specific problem or perform a set of related tasks.

- An *anonymous block* is a PL/SQL block that appears within your application and it is not named or stored in the database. In many applications, PL/SQL blocks can appear wherever SQL statements can appear.

- A *stored procedure* is a PL/SQL block that Oracle stores in the database and can be called by name from an application.

- Functions always return a single value to the caller; procedures do not return values to the caller.

- Packages are groups of procedures and functions.
CREATE PROCEDURE credit_labmark (sno NUMBER, cno CHAR, credit NUMBER) AS
    old_mark NUMBER;
    new_mark NUMBER;
BEGIN
    SELECT labmark INTO old_mark FROM enrol
    WHERE studno = sno and courseno = cno FOR UPDATE OF labmark;
    new_mark := old_mark + credit;
    UPDATE enrol SET labmark = new_mark
    WHERE studno = sno and courseno = cno;
    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO enrol(studno, courseno, labmark, exammark)
        VALUES(sno, cno, credit, null);
    WHEN OTHERS THEN  ROLLBACK;
END credit_labmark;
Function

create function get_lab_mark(sno number, cno char)
  return number
as f_lab_mark number;
  no_mark exception;
  begin
    select labmark
       into f_lab_mark from enrol
        where studno = sno and courseno = cno;
    if f_lab_mark is null
    then raise no_mark;
    else return(f_lab_mark);
    end if
  exception
    when no_mark then ....return(null);
  end;
 Stored Procedures

Created in a user's schema and stored, centrally, in compiled form in the database as a named object that can be:

- interactively executed by a user using a tool like SQL*Plus
- called explicitly in the code of a database application, such as an Oracle Forms or a Pre compiler application, or in the code of another procedure or trigger

When PL/SQL is not stored in the database, applications can send blocks of PL/SQL to the database rather than individual SQL statements → reducing network traffic.
Architecture

Oracle Server

SGA

Procedure
Begin
Procedurecall
Procedurecall
SQL
End;

PL/SQL Engine

Procedural Statement Executor

SQL

SQL Statement Executor

Database

Program code
Program code
Procedure call
Program code
Program code

Database Application

CS530 - Ian Horrocks and Robert Stevens 8
Benefits of Procedures I

- Security
  - Control data access through procedures and functions.
  - E.g. grant users access to a procedure that updates a table, but not grant them access to the table itself.

- Performance
  - The information is sent only once between database and application and thereafter invoked when it is used.
  - Network traffic is reduced compared with issuing individual SQL statements or sending the text of an entire PL/SQL block.
  - A procedure's compiled form is readily available in the database, so no compilation is required at execution time.
  - The procedure might be cached.
Benefits of Stored Procedures II

- Memory Allocation
  - Stored procedures take advantage of the shared memory capabilities of Oracle
  - Only a single copy of the procedure needs to be loaded into memory for execution by multiple users.

- Productivity
  - By designing applications around a common set of procedures, you can avoid redundant coding and increase your productivity.
  - Procedures can be written to insert, update, or delete rows from a table and then called by any application without rewriting the SQL statements necessary to accomplish these tasks.
  - If the methods of data management change, only the procedures need to be modified, not all of the applications that use the procedures.
Benefits of Procedures III

- **Integrity**
  - Stored procedures improve the integrity and consistency of your applications. By developing all of your applications around a common group of procedures, you can reduce the likelihood of committing coding errors.
  - You can test a procedure or function to guarantee that it returns an accurate result and, once it is verified, reuse it in any number of applications without testing it again.
  - If the data structures referenced by the procedure are altered in any way, only the procedure needs to be recompiled; applications that call the procedure do not necessarily require any modifications.
Packages

- A method of encapsulating and storing related procedures, functions, variables, cursors and other package constructs together as a unit in the database for continued use as a unit.

- Similar to standalone procedures and functions, packaged procedures and functions can be called explicitly by applications or users.
  - Organise routines
  - Increased functionality (e.g. global package variables can be declared and used by any procedure in the package) and
  - Increased performance (e.g. all objects of the package are parsed, compiled, and loaded into memory once).
Package manage tasks in database

- Database applications explicitly call packaged procedures as necessary.
- After being granted the privileges for the package, a user can explicitly execute any of the procedures contained in it.
- EXECUTE marks_mgmt.credit_labmark(99234,’CS 2312’,20)
- Packages offer several development and performance advantages over standalone stored procedures;
Benefits of Packages

- Encapsulation of related procedures and variables providing:
  - Better organization during the development process and for granting privileges
- Declaration of public and private procedures, variables, constants, and cursors
- Better performance
  - An entire package is loaded into memory when a procedure within the package is called for the first time in one operation, as opposed to the separate loads required for standalone procedures. When calls to related packaged procedures occur, no disk I/O is necessary to execute the compiled code already in memory.
  - A package body can be replaced and recompiled without affecting the specification. Objects that reference a package's constructs (always via the specification) need not be recompiled unless the package specification is also replaced. Unnecessary recompilations can be minimized, so in less impact on overall database performance.
Triggers vs Procedures and Packages

- Triggers are similar to stored procedures. A trigger can include SQL and PL/SQL statements to execute as a unit and can invoke stored procedures. Triggers are stored in the database separate from their associated tables.

- Procedures and triggers differ in the way that they are invoked.
  - A procedure is explicitly executed by a user, application, or trigger.
  - Triggers (one or more) are implicitly fired (executed) by Oracle when a triggering INSERT, UPDATE, or DELETE statement is issued, no matter which user is connected or which application is being used.
Retrieval: Impedance Mismatch

What happens when the query returns several rows? The host variables can only hold one value.

Oracle will only pass the first row returned by the query to the PL/SQL block (or host language program).

Re-executing the SELECT operation will only run the query again and so the first row will be selected again.

Different type systems
Different execution models
Cursors

When a query returns multiple rows a cursor must be declared to process each row returned by the query and to keep track of which row is currently being processed.

The rows returned by a query are stored in an area called the *Active Set*.

A cursor can be thought of as pointing to a row in the active set.
PROCEDURE apply_marks IS
CURSOR marks_cursor IS
   SELECT sno, cno, kind, amount FROM marks
WHERE status = 'Pending' ORDER BY time_tag FOR UPDATE OF marks;
BEGIN
   FOR marks IN marks_cursor LOOP /* implicit open and fetch */
      new_status := 'Accepted';
      IF marks.kind = 'L' THEN
         credit_labmark(marks.sno, marks.cno, marks.amount);
      ELSIF trans.kind = 'E' THEN
         credit_exammark(marks.sno, marks.cno, marks.amount);
      ELSE new_status := 'Rejected';
      END IF;
      UPDATE marks SET status = new_status
      WHERE CURRENT OF marks_cursor;
   END LOOP; COMMIT;
END apply_marks;
Embedded SQL

SQL statements placed within a program. The source program is called the *host program*, and the language in which it is written is called the *host language*. You can execute any SQL statement using *embedded SQL* statements just as if you were in SQL*Plus.

- CREATE, ALTER and DROP database tables
- SELECT, INSERT, UPDATE and DELETE rows of data
- COMMIT transactions (make any changes to the database permanent)
Embedded SQL Statements

- Embedded SQL statements incorporate DDL, DML, and transaction control statements within a procedural language program. They are used with the Oracle pre-compilers, e.g. Pro*C.

- Embedded SQL statements enable you to
  - define, allocate, and release cursors (DECLARE CURSOR, OPEN, CLOSE)
  - declare a database name and connect to Oracle (DECLARE DATABASE, CONNECT)
  - assign variable names (DECLARE STATEMENT)
  - initialize descriptors (DESCRIBE)
  - specify how error and warning conditions are handled (WHENEVER)
  - parse and execute SQL statements (PREPARE, EXECUTE, EXECUTE IMMEDIATE)
  - retrieve data from the database (FETCH).
Executable and Declarative Statements

Embedded SQL includes all the interactive SQL statements plus others that allow you to transfer data between Oracle and a host program. There are two types of embedded SQL statements:

Executable:
- used to connect to Oracle, to define, query and manipulate Oracle data, to control access to Oracle data and to process transactions. They can be placed wherever host-language executable statements can be placed.

Declarative:
- do not operate on SQL data. Use them to declare Oracle objects, communication areas and SQL variables which will be used by Oracle and your host program. They can be placed wherever host-language declarations can be placed.
Binding Variables

A host variable is prefixed with a colon (:) in SQL statements but must not be prefixed with a colon in C statements.

```sql
EXEC SQL BEGIN DECLARE SECTION;

    INT sno;
    VARCHAR cno[5];
    INT labmark;

EXEC SQL END DECLARE SECTION;

...)

EXEC SQL SELECT labmark INTO :labmark FROM enrol

    WHERE studno = :sno and courseno = :cno
```

The case of the host variable is significant when referencing them.
SELECT

INTO clause specifies the host variables which will hold the values of the attributes returned.

EXEC SQL SELECT courseno, subject
INTO :courseno, :subject
FROM course
WHERE courseno = :menu_selection;

Attributes in the staff table.

Host variable used to supply the WHERE clause with a value to base the query on. In SQL*Plus this would be done using a literal value. Pro*C allows variables to be used to specify a value. Host variables used in this way must contain a value before the SELECT statement is used.
Example

Declare any host variables
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR studname[21];
    VARCHAR cno[5];
    INT labmark;
    VARCHAR o_connect_uid[18];
EXEC SQL END DECLARE SECTION;

Include the error handlers
EXEC SQL INCLUDE sqlca;
EXEC SQL INCLUDE oraca;

Log on procedure
void Oracle_Connect(void)
{
    (void)strcpy(o_connect_uid.arr,"/@t:ora-srv:mucs7");
    o_connect_uid.len = strlen(o_connect_uid.arr);
    EXEC SQL CONNECT :o_connect_uid;
}
Connect to Oracle Server and Do the Query

main()
{
   EXEC SQL WHENEVER SQLERROR DO sqlerror()

   EXEC ORACLE OPTION (ORACA=YES);
   oraca.orastxtf = 1; Oracle_Connect(); printf("Connected to Oracle\n");

   Cursor for query
   EXEC SQL DECLARE studcursor CURSOR FOR
      SELECT s.name, e.courseno, e.labmark,
            FROM student s, enrol e WHERE s.studno = e.studno;

   Do the query
   EXEC SQL OPEN studcursor; printf("Name/Course/LabMark\n");

   Loop to fetch rows
   while (sqlca.sqlcode == 0) {
      EXEC SQL FETCH studcursor
         INTO :studname, :cno, :labmark
         printf("%s,%s,%d", studname, cno, labmark);
   }

   printf("%ld rows selected.\n",sqlca.sqlerrd[2]);
   EXEC SQL CLOSE studcursor;
   EXEC SQL COMMIT WORK RELEASE;
   exit(1);}
Examples Class
Language Connectivity

Perl DBI
What is Perl DBI

- Perl is a Scripting Language ‘Glue’ used mainly in web applications.
- DataBase Interconnectivity uses the same call to connect to any number of supported databases.
- When a DB changes the connection is still supported.
Order

- Connect
- Do
- Disconnect
Connect

- $strConnect="DBI:
  Database:
  DatabaseName:
  Server";

- DBI->connect($strConnect, User, Password’);
Example

- my $ptrDB="";
- $ptrDB=DBI->connect("DBI:mysql:2312:manmysql.man.ac.uk", 'Si', $strPasswd);
Do

- my $strSQL_Query = 'delete from users';

- $ptrDB->do($strSQL_Query);
Disconnect

- $ptrDB->disconnect;
Example (using Select) #1

sub strSelect($)
{
    my ($strQuery)=@_; // Example script to demonstrate Select
    my $ptrDB='';
    my $hData;
    my $strOutput='';

    $bWaiting=0;
    if($ptrDB=DBI->connect("DBI:mysql:$hGlobalConfig{strDBName}:$hGlobalConfig{strDBServer}","$hGlobalConfig{strPrivateUser}$hGlobalConfig{strPrivatePass}))
    {

Example (using Select) #2

```php
if($hData=$ptrDB->prepare("select $strQuery"))
{
    if(!$hData->execute)
    {
        $strOutput=undef;
    }
    else
    {
        $strOutput=$hData->fetchall_arrayref;
        $bWaiting=1;
        if(!$strOutput->[0][0])
        {
            $strOutput=undef;
            $bWaiting=0;
        }
    }
}
else
{
    $strOutput=undef;
}
```
Example (using Select) #3

```php
$ptrDB->disconnect;
}
else
{
    $strOutput=undef;
}
return($strOutput);
}```
ODBC
What is ODBC?

- ODBC is (Open Database Connectivity):
- A standard or open application programming interface (API) for accessing a database.
- SQL Access Group, chiefly Microsoft, in 1992
- By using ODBC statements in a program, you can access files in a number of different databases, including Access, dBase, DB2, Excel, and Text.
- It allows programs to use SQL requests that will access databases without having to know the proprietary interfaces to the databases.
- ODBC handles the SQL request and converts it into a request the individual database system understands.
More on ODBC

- You need:
  - the ODBC software, and
  - a separate module or driver for each database to be accessed. Library that is dynamically connected to the application.

- Driver masks the heterogeneity of DBMS operating system and network protocol.

- E.g. (Sybase, Windows/NT, Novell driver)
ODBC Architecture

```
Application

ODBC driver manager

Driver
(DBMS/OS/network)

Data Source
```
JDBC
What is JDBC?

- JDBC is: Java Database Connectivity
  - is a Java API for connecting programs written in Java to the data in relational databases.
  - consists of a set of classes and interfaces written in the Java programming language.
  - provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API.
  - The standard defined by Sun Microsystems, allowing individual providers to implement and extend the standard with their own JDBC drivers.

- JDBC:
  - establishes a connection with a database
  - sends SQL statements
  - processes the results.
JDBC vs ODBC

- ODBC is used between applications
- JDBC is used by Java programmers to connect to databases
- With a small "bridge" program, you can use the JDBC interface to access ODBC-accessible databases.
- JDBC allows SQL-based database access for EJB persistence and for direct manipulation from CORBA, DJB or other server objects
The JDBC API supports both two-tier and three-tier models for database access.

- Two-tier model -- a Java applet or application interacts directly with the database.
- Three-tier model -- introduces a middle-level server for execution of business logic:
  - the middle tier to maintain control over data access.
  - the user can employ an easy-to-use higher-level API which is translated by the middle tier into the appropriate low-level calls.
JDBC Architectures

Java Application

JDBC driver manager

JDBC/native bridge

Native driver (DBMS specific)

JDBC/ODBC bridge

ODBC Driver

JDBC Driver (DBMS Specific)

JDBC middleware (various DBMS)

DBMS
The JDBC Steps

1. Importing Packages
2. Registering the JDBC Drivers
3. Opening a Connection to a Database
4. Creating a Statement Object
5. Executing a Query and Returning a Result Set Object
6. Processing the Result Set
7. Closing the Result Set and Statement Objects
8. Closing the Connection
1: Importing Packages

//
// Program name: LecExample_1a.java
// Purpose: Basic selection using prepared statement
//

//Import packages
import java.sql.*;  //JDBC packages
import java.math.*;
import java.io.*;
import oracle.jdbc.driver.*;
2: Registering JDBC Drivers

class LecExample_1a {

public static void main (String args []) throws SQLException {

    // Load Oracle driver
    DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());

}
3: Opening connection to a Database

// Prompt user for username and password
String user;
String password;

user = readEntry("username: ");
password = readEntry("password: ");

// Connect to the local database
Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@aardvark:1526:teach", user, password);
4. Creating a Statement Object

// Query the hotels table for resort = 'palma nova'
// Please notice the essential trim
PreparedStatement pstmt =
    conn.prepareStatement("SELECT hotelname, rating FROM hotels WHERE trim(resort) = ?");
pstmt.setString(1, "palma nova");
ResultSet rset = pstmt.executeQuery();

// Print query results
while (rset.next())
    System.out.println(rset.getString(1) + " " + rset.getString(2));
7. Closing the Result Set and Statement Objects

8. Closing the Connection

// close the result set, statement, and the connection
rset.close();
pstmt.close();
conn.close();
Mapping Data Types

- There are data types specified to SQL that need to be mapped to Java data types if the user expects Java to be able to handle them.

- Conversion falls into three categories:
  - SQL type to Java direct equivalents
    - SQL INTEGER direct equivalent of Java int data type.
  - SQL type can be converted to a Java equivalent.
    - SQL CHAR, VARCHAR, and LONGVARCHAR can all be converted to the Java String data type.
  - SQL data type is unique and requires a special Java data class object to be created specifically for their SQL equivalent.

- SQL DATE converted to the Java Date object that is defined in java.Date especially for this purpose.
Other Types of Databases
Distributed DBMS #1

- Large Amounts of Data?
- Lots of Queries?
- Make the database Parallel by distributing bits of it over different systems (possible in different locations) and using a distributed DBMS to keep track of data and queries.
Distributed DBMS #2

- Advantages
  - Organisational Structure
  - Share ability
  - Availability
  - Reliability
  - Performance
  - Modular Growth

- Disadvantages
  - Complexity
  - Cost
  - Security
  - Lack of Standards
  - Lack of Experience
  - Database Design
Just what you’d expect…
- Complex Objects (records)
- Encapsulated
- Classes and Types supported
- Inheritance
- Dynamic
- Extensible Data Types (TRIPOD)
OO DBMS #2

- Advantages
  - Enriched Modelling
  - Extensible
  - More expressive querying
  - Schema evolution
  - Performance

- Disadvantages
  - No Universal Data Model
  - Lack of Experience
  - Lack of Standards
  - Competition
  - Query optimisation compromises encapsulation
  - No views
  - Poor security
Object Relational DBMS #1

- Combination of Object and Relational Databases
- Tries to have all of the strengths of both and the weaknesses of neither.
- In essence support COMPLEX Data
- AND SQL
Object Relational DBMS #2

- Advantages
  - Solves many weakness of OODBMS

- Disadvantages
  - Complexity
  - Cost
XML DBMS #1

- Web is important
- Web has semi structured data.
- Databases try to support the data structures in the form of XML.
- And queries in XML
XML DBMS #2

- Advantages
  - Integrates with HTML
  - Follows same scheme
  - Is a description / meta language.

- Disadvantages
  - Complexity
  - Cost
  - Slow
  - No Standards
  - Lack of Experience
Data Warehousing #1

- Subject Oriented
- Integrated
- Time Variant
- Non-Volatile Collection
- To support management decision making
Data Warehousing #2

- Advantages
  - High Investment Return
  - Competitive Advantage
  - Increased Productivity of decision makers

- Disadvantages
  - Underestimation of Resources
  - Required data not captured
  - Increased end-user demands
  - Data homogenisation decreases its value
  - High maintenance
  - Ownership and integration
Multimedia DBMS #1

- Relational DBs are good at Text
- With the advent of sound and images different technology is needed to assist searching.
- Multimedia DBMS look to find fragments of images, sounds, multimedia resources within a multimedia DB.
Multimedia DBMS #2

- Advantages
  - Good for Sound
  - Good for Images
  - Good for Electronic Document Management

- Disadvantages
  - Very New
  - Not good at Text searching
Background
Examples of Packages and Procedures
Create Package Specification

create package marks_mgmt (null) as

max_mark CONSTANT NUMBER := 100.00;

PROCEDURE apply_marks;

PROCEDURE enter_marks(sno number, cno char, kind char, credit number);

end marks_mgmt;
CREATE PACKAGE BODY marks_mgmt AS

new_status   CHAR(20);  /* Global variable to record status of
    transaction being applied. Used for update in enter_marks. */

PROCEDURE do_journal_entry (sno NUMBER, cno CHAR, kind CHAR) IS
    /* Records a journal entry for each marks credit applied by the
       enter_marks procedure. */
BEGIN
    INSERT INTO journal
        VALUES (sno, cno, kind, sysdate);
    IF kind = 'L' THEN new_status := 'Lab credit';
    ELSIF kind = 'E' THEN new_status := 'Exam credit';
    ELSE new_status := 'New enrolment';
    END IF;
END do_journal_entry;

CREATE PROCEDURE credit_labmark (sno NUMBER, cno CHAR, credit NUMBER) AS
old_mark NUMBER; new_mark NUMBER;
mark_overflow EXCEPTION;
BEGIN
    SELECT labmark INTO old_mark FROM enrol
    WHERE studno = sno and courseno = cno
    FOR UPDATE OF labmark;
    new_mark := old_mark + credit;
    IF new_mark <= max_mark THEN
        UPDATE enrol SET labmark = new_mark
        WHERE studno = sno and courseno = cno;
        do_journal_entry(sno, cno, L);
    ELSE RAISE mark_overflow
    ENDIF;
END;
Create Package Body II

EXCEPTION

WHEN NO_DATA_FOUND THEN
    /* Create new enrolment if not found */
    INSERT INTO enrol (studno, courseno, labmark, exammark)
        VALUES(sno, cno, credit, null);
    do_journal_entry(sno, cno, 'N');
WHEN mark_overflow THEN
    new_status := 'Mark Overflow';
WHEN OTHERS THEN
    /* Return other errors to application */
    new_status := 'Error: ' || SQLERRM(SQLCODE);
END credit_labmark;

CREATE PROCEDURE credit_exammark (sno NUMBER, cno CHAR,
    credit NUMBER) AS...

END credit_exammark;
PROCEDURE apply_marks IS .... complete shortly...
END apply_marks;

PROCEDURE enter_marks(sno NUMBER, cno CHAR, kind CHAR, credit NUMBER) IS

/* A new mark is always put into this 'queue' before being applied to the specified enrolment instance by the APPLY_MARKS procedure. */
BEGIN
    INSERT INTO marks
    VALUES (sno, cno, kind, amount, 'Pending', sysdate);
    COMMIT;
END enter_marks;

END marks_mgmt ; /* end package */
Additional Material
void sqlerror(void)
{
    int o_errl;
    int len = 550;
    char o_err[550];

    EXEC SQL WHENEVER SQLERROR CONTINUE;
    sqlca.sqlerrm.sqlerrmc[sqlca.sqlerrm.sqlerrml] = NULL;
    printf("\nOracle Error:\n%s", sqlca.sqlerrm.sqlerrmc);

    oraca.orastxt.orastxtc[oraca.orastxt.orastxtl] = NULL;
    printf("ERROR statement:%s\n", oraca.orastxt.orastxtc);

    sqlglm(o_err, &len, &oerrl);
    o_err[o_errl] = NULL;
    printf("ERROR Details: %s\n", o_err);

    oraca.orasfnm.orasfnmc[oraca.orasfnm.orasfnml] = NULL;
    printf("ERROR at line %ld in %s\n",
            oraca.oraslnr, oraca.orasfnm.orasfnmc);
    EXEC SQL ROLLBACK WORK RELEASE;
    exit(0);}

An error handling procedure
SQLJ
What is SQLJ?

- SQLJ is a set of programming extensions that allow a programmer using the Java programming language to embed statements that provide SQL database requests.
- SQLJ is similar to existing extensions for SQL that are provided for C, FORTRAN, and other programming languages.
- IBM, Oracle, and several other companies are proposed SQLJ as a standard and as a simpler and easier-to-use alternative to JDBC.
SQLJ Specifications

- The SQLJ specifications are in several parts:
  - SQLJ: Embedded SQL...Specifications for embedding SQL statements in Java methods.
  - SQLJ: SQL Routines...Specifications for calling Java static methods as SQL stored procedures and user-defined functions.
  - SQLJ: SQL Types...Specifications for using Java classes as SQL user-defined data types.
SQLJ Example

```java
String bug = "spider";
#sql {
    INSERT INTO bugs (name, numLegs)
    VALUES (:bug, :
        (getNumLegs(bug)))
};
```

SQL can span multiple lines
Java host expressions in SQL statement
`throws java.sql.SQLException`
JDBC Example

```java
PreparedStatement stmt = conn.createStatement(
    "INSERT INTO bugs (name, numLegs) VALUES
    (?, ?)"");
stmt.setString(1, bug);
stmt.setInt(2, getNumLegs(bug));
stmt.executeUpdate();
stmt.close();
```

JDBC needs:

- explicit statement handles
- explicit `setXxx` binds
- explicit connection
### SQLJ vs JDBC comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>SQLJ</th>
<th>JDBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL statements</td>
<td>static</td>
<td>dynamic</td>
</tr>
<tr>
<td>Strong typing</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Checking</td>
<td>static</td>
<td>runtime only</td>
</tr>
<tr>
<td>Syntax</td>
<td>concise</td>
<td>API</td>
</tr>
<tr>
<td>Standard</td>
<td>ANSI</td>
<td>Sun</td>
</tr>
<tr>
<td>Portable</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Object support</td>
<td>yes*</td>
<td>yes*</td>
</tr>
</tbody>
</table>
Use SQLJ to write your program when

- you want to be able to check your program for errors at translation-time rather than at run-time.
- you want to write an application that you can deploy to another database. Using SQLJ, you can customize the static SQL for that database at deployment-time.
- you are working with a database that contains compiled SQL. You will want to use SQLJ because you cannot compile SQL statements in a JDBC program.
Use JDBC to write your program when

- your program uses dynamic SQL. For example, you have a program that builds queries on-the-fly or has an interactive component.

- you do not want to have a SQLJ layer during deployment or development. For example, you might want to download only the JDBC Thin driver and not the SQLJ runtime libraries to minimize download time over a slow link.
SQLJ static and non-static SQL

- The standard covers only *static SQL* operations
  - those that are predefined and do not change in real-time as a user runs the application
  - of course the data values that are transmitted can change dynamically!

- Oracle SQLJ offers extensions to support *dynamic SQL* operations
  - those that are *not* predefined, where the operations *themselves* can change in real-time.

- It is possible to use dynamic SQL operations through JDBC code or PL/SQL code within a SQLJ application.

- Typical applications contain much more static SQL than dynamic SQL.
Java and SQLJ versus PL/SQL

- Java and PL/SQL are complementary.
- Suited for different kinds of applications.

- PL/SQL is better for SQL-intensive applications.
  - Optimized for SQL, and so SQL operations are faster in PL/SQL than in Java.
  - Uses SQL datatypes directly, while Java applications must convert between SQL datatypes and Java types.

- Java, is better for logic-intensive applications.
  - Superior programming model.
  - Java's more general type system is better suited than PL/SQL for component-oriented applications.
Interoperability: SQLJ and PL/SQL

- PL/SQL programs
  - transparently call Java stored procedures, enabling you to build component-based Enterprise JavaBeans and CORBA applications.
  - have transparent access to a wide variety of existing Java class libraries through trivial PL/SQL call specifications.

- Java programs
  - call PL/SQL stored procedures and anonymous blocks through JDBC or SQLJ.
  - SQLJ provides syntax for calling stored procedures and functions from within a SQLJ statement, and also supports embedded PL/SQL anonymous blocks within a SQLJ statement.
Additional Material
// Method:  readEntry
// Purpose: to read a string from the user and return it
// Input:   The prompt string
// Output:  User entry

static String readEntry (String prompt)
{
    try{
        StringBuffer buffer = new StringBuffer ();
        System.out.print (prompt);
        System.out.flush ();
        int c = System.in.read ();
        while (c != '\n' && c != -1){
            buffer.append ((char)c);
            c = System.in.read ();
        }
        return buffer.toString ().trim ();
    }
    catch (IOException e){
        return "";
    }
    return "";
}