>> Accessing SQL From a Programming Language

>> API

>> calls to:
Connect to DB
Send SQL commands
Fetch results

>> approaches:
Dynamic SQL: in run time
JDBC (Java Database Connectivity)
ODBC (Open Database Connectivity)
Embedded SQL: in compile time

>> Dynamic SQL:
>> JDBC:
a java API that support SQL --> for query, update, retrieving and ...

by these models:
Open a connection
Create a “statement” object
Execute queries using the Statement object to send queries and fetch results
Exception mechanism to handle errors

>> Database Connection:
JDBC driver that must be dynamically loaded to access the database from Java
This is done by invoke:
Class.forName("oracle.jdbc.driver.OracleDriver");
implementing the java.sql.Driver interface

>> Connecting to the Database: open connection
getConnection method of DriverManager class
Connection conn = DriverManager.getConnection
("Server URL", "User ID", "Password");

>> Methods for executing a statement:
>> executeQuery: when statement is a query and it has a result
>> executeUpdate: when statement is nonquery and it hasn't result --> Update, Insert, Delete, Create Table
It returns number of tuples
or
return zero in DDL statements

```
                        >> Retrieving the Results
                                Retrieving the set of tuples in the result into a
ResultSet object
                                Fetching the results one tuple at a time
                                Using the next method on the result set to test
whether there remains unfetched tuple in the result

                                >> Attributes are retrieved using various methods
names begin with get
                                        getString: retrieve basic SQL data types
                                        getFloat

                                                >> Possible argument get methods
                                                        attribute name as a string
                                                        An integer indicating the
position of attribute within tuple

                        # The statement and connection are both closed at the end
of the Java program because there is a limit on number of connections

                        >> Prepared Statements:
                                Creating a prepared statement which some values
replaced by “?”
                                actual values will be provided later
                                Compiling query when it is prepared

                                        PreparedStatement pStmt =
conn.prepareStatement
                                                                        ("insert
into instructor values(?,?)");
                                        pStmt.setString(1, "88877");
                                        pStmt.setString(2, "Perry");
                                        pStmt.executeUpdate();
                                        pStmt.setString(1, "88878");
                                        pStmt.executeUpdate();

                        >> Metadata Features:
                                Capturing metadata about: 1. Database 2. ResultSet
(relations)
                                        ResultSetMetaData rsmd = rs.getMetaData();
                                        for(int i = 1; i <= rsmd.getColumnCount();
i++){

System.out.println(rsmd.getColumnName(i));

System.out.println(rsmd.getColumnTypeName(i));
                                        }

                >> ODBC: Open DataBase Connectivity (ODBC) standard
```

open a connection with a database, send queries and updates, get back results.

>> Embedded SQL

EXEC SQL statement use to request preprocessor

EXEC SQL <embedded SQL statement >;

#

In some languages, like COBOL, the semicolon is replaced with END-EXEC

In Java embedding uses

SQL { …. };

>> Database Connection

EXEC-SQL connect to server user user-name using password;

>> Variables

Variables of host language can use in embedded SQL statements.

preceded by a colon (:) to distinguish from SQL variables (:credit_amount)

Variables must be declared in DECLARE section

```
EXEC-SQL BEGIN DECLARE SECTION;
        int credit-amount;
EXEC-SQL END DECLARE SECTION;
```

>> SQL Query

declare c cursor for <SQL query>

c is used to identify the query

```
EXEC SQL
        declare c cursor for
        select ID, name
        from student
        where tot_cred > :credit_amount
END_EXEC
```

>> open statement: again execute the query:

```
EXEC SQL open c;
```

>> fetch statement: Placing the values into host language variables

```
EXEC SQL
        fetch c into :si, :sn

        END_EXEC
```

>> close statement: delete the temporary relation that holds the result of the query

```
                        EXEC SQL close c;

                >> update:
                        EXEC SQL < any valid update, insert, or delete>;


                        EXEC SQL
                                declare c cursor for
                                        select *
                                        from instructor
                                        where dept_name = 'Music'

                                        for update;

                        EXEC SQL
                                update instructor
                                        set salary = salary + 1000

                                        where current of c;
```

>> Functions and Procedures: Functions/procedures can be written in SQL, or in an external programming language

```
        create function dept_count (dept_name varchar(20))
                returns integer
                begin
                        declare d_count integer;
                                select count (* ) into d_count

                                from instructor
                                where instructor.dept_name = dept_name

                        return d_count;
                end

        select dept_name, budget
        from department
        where dept_count (dept_name) > 12

        # Compound statement: begin … end
        # returns: variable-type that returned
        # return: values that are returned as result

        >> Table Functions: functions that return a relation as a result
                create function instructor_of (dept_name char(20))
                        returns table (
                                ID varchar(5),
                                name varchar(20),
                                dept_name varchar(20),
                                salary numeric(8,2))
                        return table
```

```
                                (select ID, name, dept_name, salary
                                from instructor
                                where instructor.dept_name =
instructor_of.dept_name)

                select *
                from table (instructor_of ('Music'))
```

>> Triggers: a statement that executed automatically by system as a side effect of modification to database

we must:

1. Specify conditions which the trigger is to be executed.
2. Specify actions to be taken when trigger executes.

events: insert, delete or update

Triggers on update can be restricted to specific attributes

```
after update of takes on grade
```

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
```

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

For example: convert blank grades to null.

>> Using set Statement

```
create trigger setnull before update on takes
referencing new row as nrow
for each row
when (nrow.grade=' ')
begin atomic
        set nrow.grade=null;
end;
```

>> Maintain Referential Integrity

```
create trigger timeslot_check1 after insert on section
referencing new row as nrow
for each row
when (nrow.time slot_id not in (
        select time slot_id
        from time slot))
begin
        rollback
end;

create trigger timeslot_check2 after delete on timeslot
referencing old row as orow
for each row
```

```
                when (orow.time slot id not in (
                                select time slot_id
                                from time slot)
                        and orow.time_slot_id in (
                                select time_slot_id
                                from section))
                begin
                        rollback
                end;

        >>  Maintain credits_earned value
                create trigger credits_earned after update of takes on (grade)
                referencing new row as nrow
                referencing old row as orow
                for each row
                when nrow.grade <> 'F' and nrow.grade is not null
                        and (orow.grade='F' or orow.grade is null)
                begin atomic
                        update student
                        set tot cred=tot cred+
                                (select credits
                                from course
                                where course.course_id=nrow.course_id)
                        where student.id = nrow.id;
                end;

        >> Triggers can be disabled by (default is enable):
                alter trigger trigger_name disable
                disable trigger trigger_name

        >> A trigger can be dropped
                drop trigger trigger_name

        >> Triggers were used for: 1. maintain summary, 2. support for replication
                and now DBs havr built-in support for these

                 Encapsulation facilities can be used instead of triggers in cases
:
                        1.Define methods to update fields
                        2. Carry out actions as part of the update methods instead
of through a trigger

        >>  Risk of unintended execution of triggers
                        for example, when
                                * Loading data from a backup copy
                                * Replicating updates at a remote site
                                * Error leading to failure of critical transactions
that set off the trigger
                                * Cascading execution
                        Trigger execution can be disabled before such actions.
```

```
        >> Ranking
                select ID, rank() over (order by GPA desc) as s_rank
                from student_grades
                order by s_rank

                >> Ranking with Partitions
                        select ID, dept_name, rank () over (partition by dept_name
order by GPA desc)
                                                                as dept_rank
                        from dept_grades
                        order by dept_name, dept_rank;

                # Ranking is done after applying group by
                # limit n: Top n Items
                # ntile: takes the tuples in each partition and divides them into n
buckets
                        select ID, ntile(4) over (order by GPA desc) as quartile
                        from student_grades;

                # nulls first or nulls last
                        select ID, rank ( ) over (order by GPA desc nulls last) as
s_rank
                        from student_grades
```