

COMP9120

Week 8: Database Application Development

Semester 2, 2016

(Ramakrishnan/Gehrke – Chapter 6; 7.5

Kifer/Bernstein/Lewis – Chapter 8;

Ullman/Widom – Chapter 9)

Based on material by Dr. Bryn Jeffries
And Dr. Uwe Röhm



THE UNIVERSITY OF
SYDNEY



COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Redundant Data and Functional Dependencies

SID	first	last	dept	advisor	award	description
1234	Homer	Simpson	IT	Codd	Rejected	Work not deemed sufficient
3456	Albert	Einstein	IT	Boyce	Conditional	Accepted with minor corrections
3456	Albert	Einstein	Physics	Newton	Accepted	Accepted with no corrections
7546	Alan	Turing	IT	Codd	Accepted	Accepted with no corrections
4879	Brian	Cox	Physics	Newton	Conditional	Accepted with minor corrections
4879	Brian	Cox	Media	Attenborough	Accepted	Accepted with no corrections

sid → first, last

advisor → dept

award → description



Redundant Data and Functional Dependencies

SID	first	last	dept	advisor	award	description

sid \rightarrow first, last

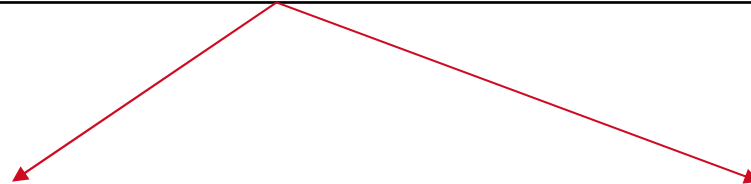
advisor \rightarrow dept

award \rightarrow description



Redundant Data and Functional Dependencies

SID	first	last	dept	advisor	award	description



SID	first	last	dept	advisor	award

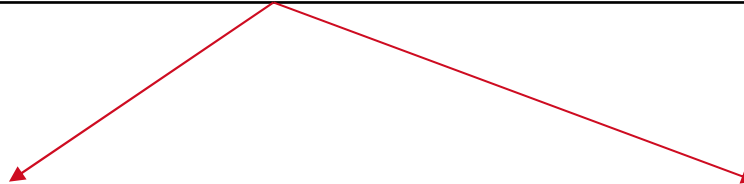
award	description

award → description



Redundant Data and Functional Dependencies

SID	first	last	dept	advisor	award	description



SID	first	last	dept	advisor	award

award	description

Dependency Preserving Property?

$\text{sid} \rightarrow \text{first, last}$

$\text{advisor} \rightarrow \text{dept}$

$\text{award} \rightarrow \text{description}$

(If Y is a subset of X)



1. Reflexivity rule: If $Y \subseteq X$, then $X \rightarrow Y$

$A, B \rightarrow A$

$A, B \rightarrow B$

$A, B \rightarrow A, B$

- › Keywords
 - Cursors
 - Prepared Statement
 - Stored Procedure
 - SQL Injection
 - › Understanding (within a DB application API)
 - NULL handling
 - Error handling
 - › Skills
 - Write application code (e.g. Java functions with JDBC) to interact with a database
 - Identify and avoid major security flaws in client code
 - Write stored procedures and call from client code
-



THE UNIVERSITY OF
SYDNEY

Database Applications

Online Banking

Locations · Help · Mail · Sign Off

Balance Sheet

Transfer

Interest

Bill Payer

Quick Pay

Recurring Payments

Pending Payments

Payment History

Add Merchant/Payee

Delete Merchant

Customer Service

Myaccess Checking

Account: Select Account

Printable View

Summary

Account: 12346578

Balance: \$768.00

Available Balance: \$764.00

Personal schedule of fees

Go To: Past 30 days · · ·

Date	Description	Withdrawal	Deposit	Balance
05/01/09	BILL PAYER (PC) 1234567890123456 BK OF A MC	-\$1.00		\$0.00
05/01/09	ONLINE BANKING TRNSFR TO SAVINGS 34567890	-\$100.00		\$1.00
04/10/09	BILL PAYER (PC) 5289000123456789 BK OF A MC	-\$125.00		\$0.00
04/07/09	TRANSFER FROM CREDIT CARD		\$10.00	\$125.00
03/27/09	MONTHLY SERVICE CHARGE	-\$5.95		\$115.00

Go To: Past 30 days · · ·

Accession Lot: 99/85 Museum product - Museum of Applied Arts and Sciences - (1981 -) - 10/8/1999

Credit Line:

Other Details

Short Title: Chair made by Peter Stevenson, 1998

Description: Dining chair and design drawing, reproduction of original used in Cafe Australia, wood / leather / wool / horsehair / paper, designed by Marion Mahony Griffin, Melbourne, Vic Stevenson, Powerhouse Museum workshop, Museum of Applied Arts and Sciences, Sydney, Australia, 1998

Reproduction dining chair made from original design used in Cafe Australia. The chair is made of stained blackwood. The front of the backrest is upholstered in leather between which converge at the back of the seat. The circular drop-in, padded seat is upholstered in leather above a curved apron and rests on four splayed legs. A small shelf with a leather top is under the seat. The chair comes complete with design drawing used to make the reproduction.

Current Location: SPLIT - See parts for locations

Date Moved: 14/08/2008

Summary Description Production History Significance Dimensions Location Multimedia Parts Authority Related References

Display Object 1 of 1

Book - Flights - Flights - Australia

QANTAS
Spirit of Australia

Search

Home Plan Book Fly Frequent Flyer Business Solutions About Qantas Help

Flights Hotels Cars Transfers Activities Packages Shows & Events Travel Insurance Gift Vouchers Price Promise Manage Your Booking

Flights

Domestic Flights

International Flights

Multi-Stop Flights

Groups

Conferences

Conditions of Carriage

Fare Types

Toolbar

RSS Feeds

Flights

1. Search 2. Select 3. Review 4. Passengers 5. Payment 6. Confirmation

Flights from Australia (Change)

Your past flight searches

☐ Return ☐ One Way ☐ Multi

From: Sydney

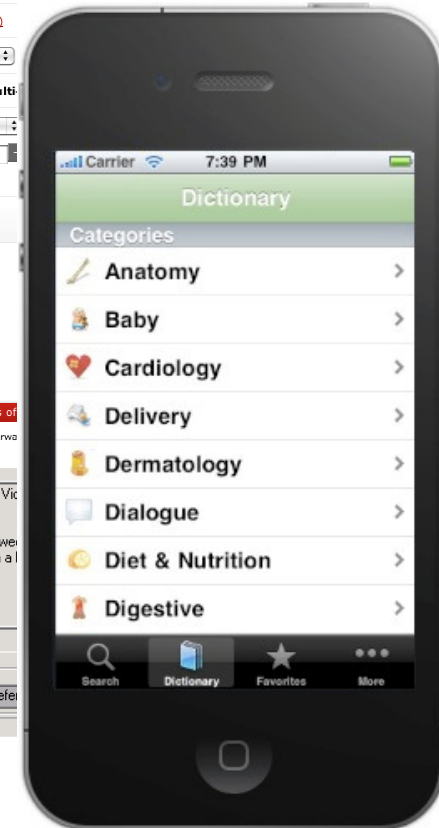
To: Melbourne

Adults: 1

Enjoy extra legroom with an Exit Row Seat

Contacts | Privacy & Security | Terms of Service

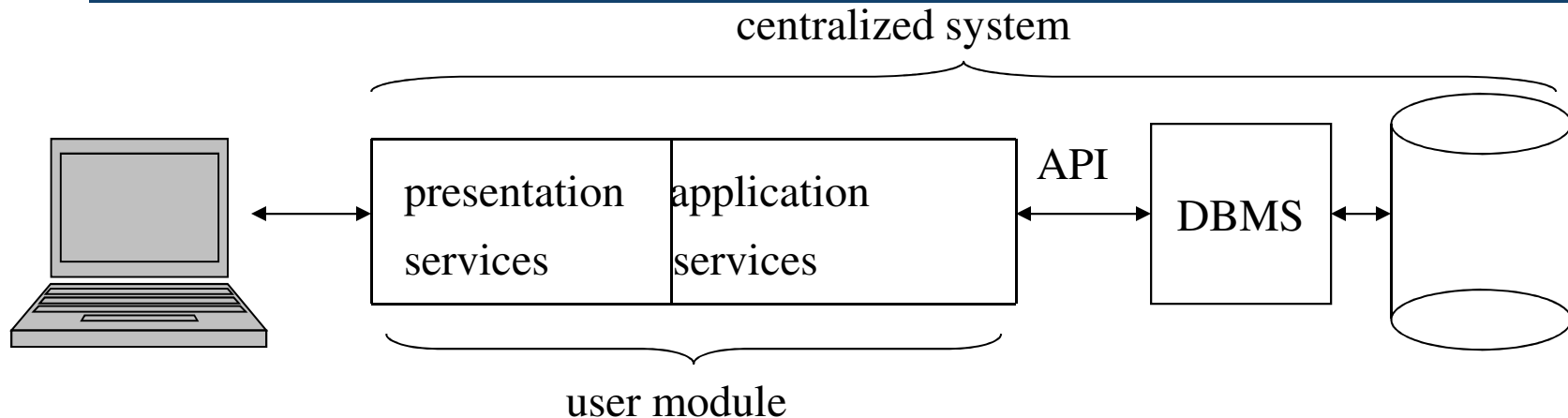
Qantas Airways



Interactive vs. Non-Interactive SQL

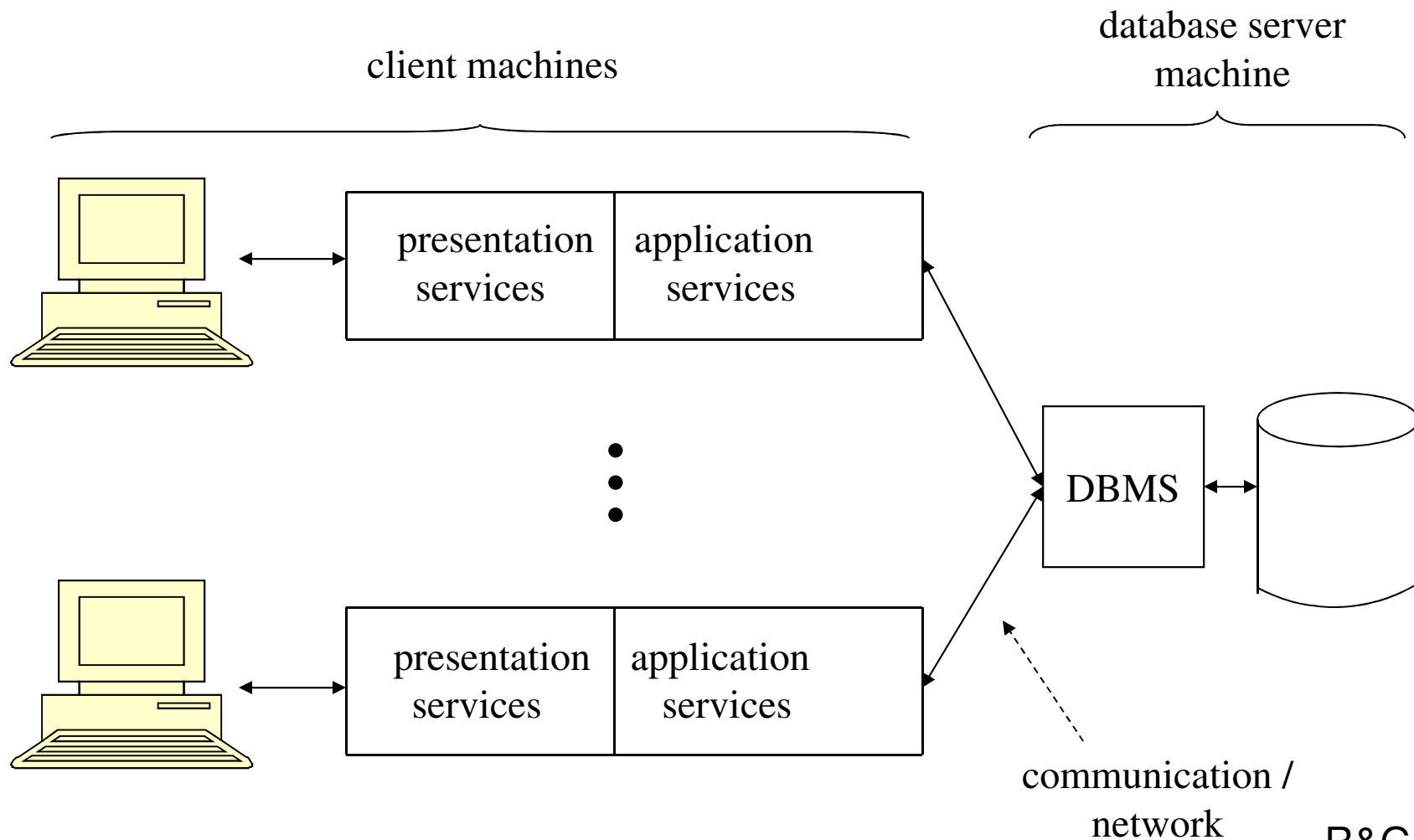
- › *Interactive SQL*: SQL statements input from terminal; DBMS outputs to screen
 - Inadequate for most uses
 - It may be necessary to process the data before output
 - Amount of data returned not known in advance
 - SQL has very limited expressive power (not Turing-complete)
 - › *Non-interactive SQL*: SQL statements are included in an application program written in a host language, like C, Java, Python, PHP
 - › *Client-side vs. Server-side* application development
 - Server-side: Stored Procedures and Triggers
-

1-Tier Architecture: Centralized System

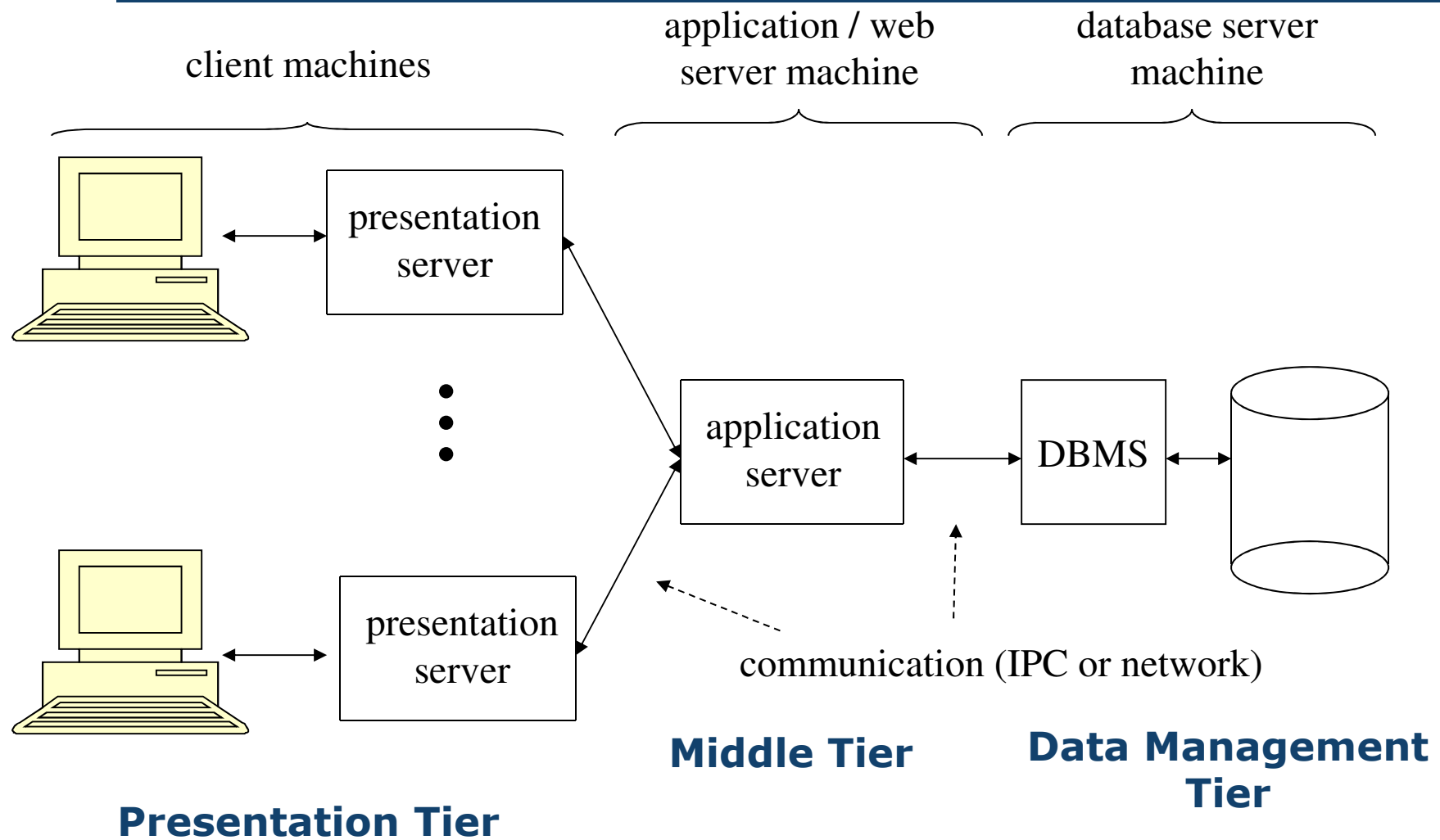


- *Presentation Services* - displays forms, handles flow of information to/from screen
- *Application Services* - implements user request, interacts with DBMS
- *DBMS* handles user requests – retrieves and returns results of queries or handles user requests for inserts/updates/deletes of data.
- Examples: Any application with integrated DB
 - ▶ MS Access systems,
 - ▶ SQLite, esp. Smartphone apps

2-Tier Architecture: Client - Server Model



Three-Tiered Architecture



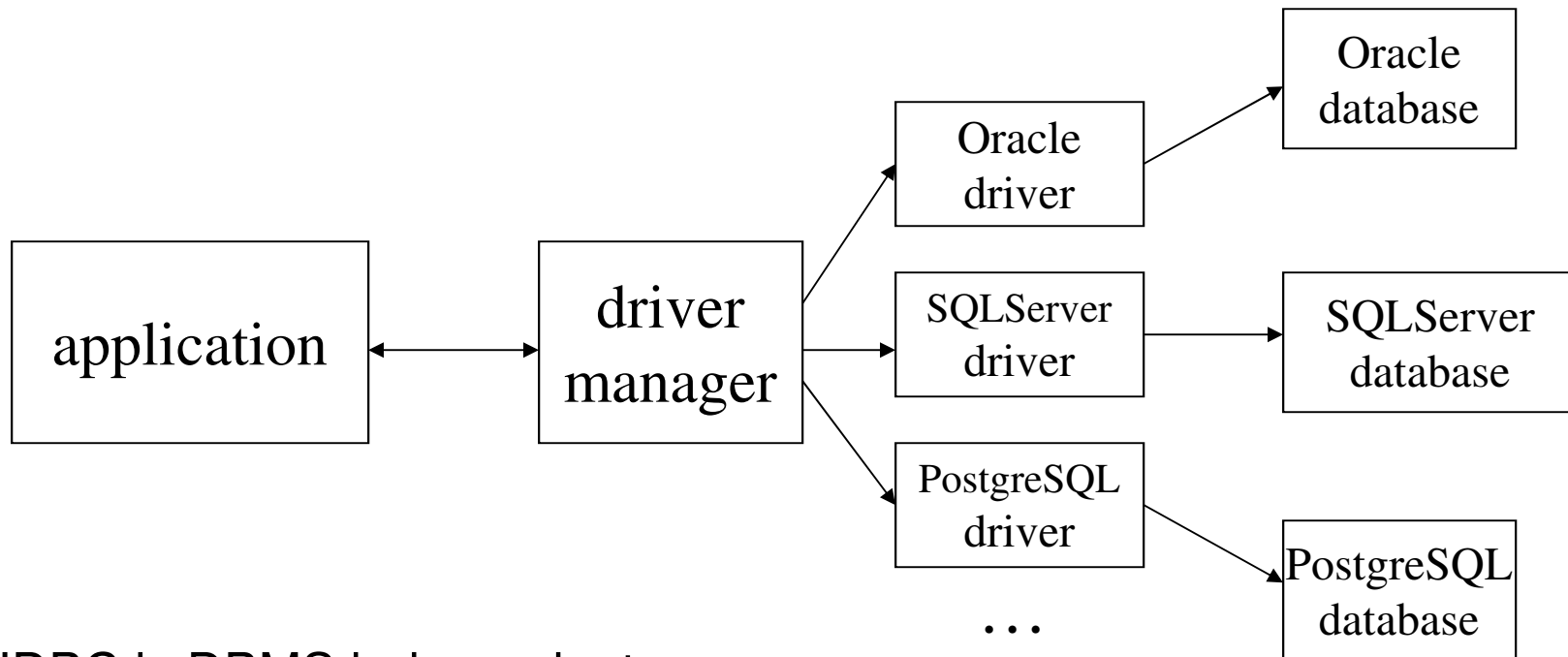
- SQL commands can be called from within a *host language* (e.g., C++, Java, PHP, Python) program.
 - ▶ SQL statements can refer to *host variables* (including special variables used to return status).
 - ▶ Must include a statement to *connect* to the right database.
- Two main integration approaches:
 - ▶ *Statement-level interface (SLI)*
 - Embed SQL in the host language (Embedded SQL in C, SQLJ)
 - Application program is a mixture of host language statements and SQL statements and directives
 - ▶ *Call-level interface (CLI)*
 - Create special API to call SQL commands (ODBC, JDBC, PHP-PDO, etc.)
 - SQL statements are passed as arguments to host language (library) procedures / APIs

```
boolean myFunction(int myValue, String myText) {  
    boolean valueToReturn = false;  
    try {  
        // Do something with input parameters  
  
        // ** Interact with database **  
  
        // Set return value  
  
    } catch (Exception e) {  
        // Handle errors  
    } finally {  
        // Cleanup  
    }  
    return valueToReturn;  
}
```

JDBC - “Java Database Connectivity”

- › JDBC is a **Java API** for communicating with database systems supporting SQL
 - › JDBC supports a variety of features for querying and updating data, and for retrieving query results
 - › JDBC also supports 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:
 - Acquire a connection
 - Create a “statement” object
 - Execute queries using the Statement object to send queries and fetch results
 - Exception mechanism to handle errors
-

JDBC Run-Time Architecture

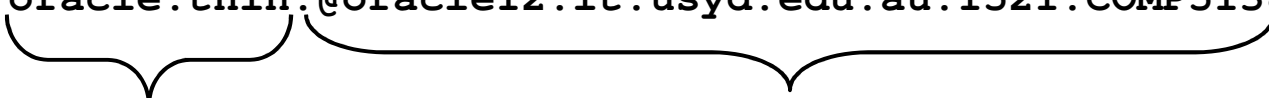


- › JDBC is DBMS independent
 - JDBC functions are generic
 - DriverManager allows to connect to specific driver
 - Even to different databases from the same program
 - Database drivers are loaded and used at run-time
-

JDBC Connection: Connecting to a database

- › A session with a data source is started through the creation of a Connection object
 - Can do this Via the `java.sql.DriverManager`:
`DriverManager.getConnection(DB_URL, userid, passwd) ;`
 - Release resource with `conn.close()` ;
 - Takes time, so try to re-use between functions
 - Pooling: <http://docs.oracle.com/javase/tutorial/jdbc/basics/sqldatasources.html>
 - Database URL of the form
 - **`jdbc:<subprotocol>:<connectionParameters>`**
- › For example with Oracle:

```
Connection conn = DriverManager.getConnection(  
    "jdbc:oracle:thin:@oracle12.it.usyd.edu.au:1521:COMP5138", user, pwd);
```


subprotocol *connectionParameters*

JDBC Statement: Executing a query

- › SQL operations are conducted using `java.sql.Statement`

- Constructed from a Connection object:

```
Statement stmt = conn.createStatement();
```

- Execute a SQL query:

```
stmt.executeQuery("SELECT ... ");
```

- Execute a DML statement (INSERT/UPDATE/DELETE)

```
stmt.executeUpdate("INSERT INTO ... ");
```

- Release resource with `stmt.close();`

- › Two other ways (covered later)

- *PreparedStatement* (semi-static SQL statements)
 - *CallableStatement* (stored procedures)
-

JDBC ResultSet: Retrieving results

- › Statement.executeQuery returns data, encapsulated in a ResultSet object (a cursor)

```
ResultSet rs = stmt.executeQuery(sql);
```

```
while (rs.next()) { // Iterate through records  
    // process the data with, e.g. rs.getInt("name");  
}
```

```
rs.close() // Release resources
```

- › A ResultSet can be a very powerful cursor:
 - previous(): moves one row back
 - absolute(int num): moves to the row with the specified number
 - relative (int num): moves forward or backward
 - first() and last(): jump to ends
 - wasNull(): dealing with NULL values
 - Support for updatable ResultSet
-

Matching Java and SQL Types

SQL Type	Java class	ResultSet get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()



Exercise: Simple Java/JDBC function

Complete the following function to list the ids of all students from the relation *Student* (studId, name, password, address)

```
void listStudentIds() throws SQLException {  
    // Connect to the database  
  
    // Execute query  
  
    // Process results  
    while ( )  
    {  
        System.out.println(  
    }  
  
    // Cleanup  
  
}
```




THE UNIVERSITY OF
SYDNEY

Time for a Break...



User Name:	' or '1'='1
Password:	' or '1'='1
Details:	

<http://www.djekldevelopments.co.uk/microsoft-visual-basic-net-programmers-cookbook/source/6797final/lib0331.html>

Should use salted hashes to avoid retrieval of plain text passwords

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(
    "SELECT * FROM Student WHERE
    name = '" + uName + "' AND Pass = '" + uPass + "'");
```

Will return all students' details!!

```
SELECT * FROM Student WHERE
Name = '' or '1'='1' AND Pass = '' or '1'='1'
```


User Name:	' or '1'='1
Password:	' or '1'='1
Details:	

- › How do we limit risks?
 - Hide error messages that expose the internals
 - Use salted hash passwords
 - Prevent SQL injection with Prepared Statements
 - Restrict user access to DB Objects (Access Control)
-

INVALID SQL: 1016 : Can't open file: 'xcart_products.MYI'. (errno: 145)
SQL QUERY FAILURE: **SELECT** COUNT(*) FROM xcart_products WHERE xcart_products.forsale='Y' and (categoryid='290' OR categoryid1='290' OR categoryid2='290' OR categoryid3='290')
INVALID SQL: 1016 : Can't open file: 'xcart_products.MYI'. (errno: 145)
SQL QUERY FAILURE: **SELECT** COUNT(*) FROM xcart_products WHERE xcart_products.forsale='Y' and (categoryid='300' OR categoryid1='300' OR categoryid2='300' OR categoryid3='300')

INVALID SQL: 1016 :
SQL QUERY FAILURE:
INVALID SQL: 1016 :
SQL QUERY FAILURE:
INVALID SQL: 1016 :
SQL QUERY FAILURE:



Association for Computing Machinery
 Advancing Computing as a Science & Profession

ACM Order Rectification

The web site you are accessing has experienced an unexpected error.
 Please contact the website administrator.

The following information is meant for the website developer for debugging purposes.

Error Occurred While Processing Request

Element ORDERID is undefined in URL.

The error occurred in **D:\wwwroot\Public\rectifyCC\rectifyCC.cfm: line 463**

```
461 : WHERE a.order_id = b.order_id
462 : AND a.order_id = c.order_id
463 : AND a.order_id = '#URL.orderID#'
464 : </CFQUERY>
465 :
```

Resources:

- Check the [ColdFusion documentation](#) to verify that you are using the correct syntax.
- Search the [Knowledge Base](#) to find a solution to your problem.

Browser Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_8; en-us) AppleWebKit/531.9
 (KHTML, like Gecko) Version/4.0.3 Safari/531.9

Remote 129.78.220.7



Cauta:

JDBC SQLException: Handling Errors

- › Most of java.sql can throw an **SQLException** if an error occurs.
 - Catch and process with `catch (SQLException e) { ... }`
 - Use `getMessage()` or `getSQLState()` or `getErrorCode()` to identify problem
 - › Sub-classes available to catch specific types e.g.:
 - `SQLTransientException` (worth retrying)
 - `SQLRecoverableException` (worth retrying with new connection)
 - `SQLTimeoutException`
 - `SQLIntegrityConstraintViolationException`
 - › **SQLWarning** is a subclass of `SQLException`; not as severe
 - `con.getWarnings();`
 - `con.getNextWarning();`
 - `con.clearWarnings();`
-

JDBC SQLException: Handling Errors

- › Most of java.sql can throw an **SQLException** if an error occurs.
 - Catch and process with `catch (SQLException e) { ... }`
 - Use `getMessage()` or `getSQLState()` or `getErrorCode()`

Table 12. Class Code 23: Constraint Violation

SQLSTATE Value	Meaning
23001	The update or delete of a parent key is prevented by a RESTRICT update or delete rule.
23502	An insert or update value is null, but the column cannot contain null values.
23503	The insert or update value of a foreign key is invalid.
23504	The update or delete of a parent key is prevented by a NO ACTION update or delete rule.
23505	A violation of the constraint imposed by a unique index or a unique constraint occurred.
23510	A violation of a constraint on the use of the command imposed by the RLST table occurred.
23511	A parent row cannot be deleted, because the check constraint restricts the deletion.
23512	The check constraint cannot be added, because the table contains rows that do not satisfy the constraint definition.
23513	The resulting row of the INSERT or UPDATE does not conform to the check constraint definition.
23514	Check data processing has found constraint violations.
23515	The unique index could not be created or unique constraint added, because the table contains duplicate values of the specified key.
23520	The foreign key cannot be defined, because all of its values are not equal to a parent key of the parent table.
23521	The update of a catalog table violates an internal constraint.

<ftp://ftp.software.ibm.com/ps/products/db2/info/vr6/htm/db2m0/db2state.htm#HDR>

STTMSG

JDBC SQLException: Handling Errors

- › Most of java.sql can throw an **SQLException** if an error occurs.
 - Catch and process with `catch (SQLException e) { ... }`
 - Use `getMessage()` or `getSQLState()` or `getErrorCode()` to identify problem
 - › Sub-classes available to catch specific types e.g.:
 - `SQLTransientException` (worth retrying)
 - `SQLRecoverableException` (worth retrying with new connection)
 - `SQLTimeoutException`
 - `SQLIntegrityConstraintViolationException`
 - › **SQLWarning** is a subclass of `SQLException`; not as severe
 - `con.getWarnings();`
 - `con.getNextWarning();`
 - `con.clearWarnings();`
-

The following function may fail to work for a number of reasons.

```
void exampleEnrolment() {  
    Connection conn = DriverManager.getConnection(  
        "jdbc:oracle:thin:@" +  
        "oracle12.it.usyd.edu.au:1521:COMP5138",  
        "COMP5138_DEMO", "COMP5138_DEMO");  
  
    Statement stmt = conn.createStatement();  
    stmt.executeUpdate("INSERT INTO Transcript VALUES  
        (309187546, 'COMP5338', 'S2', 2013, 'HD')");  
  
    conn.close();  
}
```

May fail due to
server/connection
problems

- Query could time out
- Primary key violation
- Foreign key violation

```

void exampleEnrolment() {
    Connection conn = null;
    try {
        conn = DriverManager.getConnection("jdbc:oracle:thin:@" +
            "oracle12.it.usyd.edu.au:1521:COMP5138",
            "COMP5138_DEMO", "COMP5138_DEMO");

        Statement stmt = conn.createStatement();
        stmt.executeUpdate("INSERT INTO Transcript VALUES
            (309187546, 'COMP5338', 'S2', 2013, 'HD')");
    } catch (SQLIntegrityConstraintViolationException e) {
        System.err.println("Violated a constraint!");
    } catch (SQLTimeoutException e) {
        System.err.println("Operation timed out");
    } catch (SQLException e) {
        if (e.getSQLState().equalsIgnoreCase("0A000")) {
            System.err.println("Feature not supported");
        } else {
            System.err.println("Other problem");
        }
    }
} finally {
    if (conn != null)
        try {conn.close();} catch (SQLException e) {//handle exception}
}
}

```

Dynamic SQL with Prepared Statements in JDBC

- › `java.sql.PreparedStatement` allows a statement to be executed with host variables after the query plan has been evaluated

- › Example:

```
credit (int amount, int accountno) {  
    stmt = conn.prepareStatement("UPDATE account  
                                SET balance = balance + ?  
                                WHERE account_number = ?");  
  
    stmt.setFloat(1, amount);  
  
    stmt.setInteger(2, accountno);  
  
    result = stmt.executeQuery();  
}
```

- › More flexible and secure
 - › As a rule, always use in preference to Statements
-

Another example : Security

- > What security issues can you identify in this function? How could they be fixed?

```
void getStudentAddress(int studID, String password) {  
    Connection conn = DriverManager.getConnection(  
        "jdbc:oracle:thin:@" +  
        "oracle12.it.usyd.edu.au:1521:COMP5138",  
        "COMP5138_DEMO", "COMP5138_DEMO");  
  
    Statement stmt = conn.createStatement();  
    ResultSet rs = stmt.executeQuery(  
        "SELECT address FROM Student WHERE studId='"  
        + studID + "' AND password = '" + password + "'");  
  
    // Process results  
    while (rs.next()) {  
        System.out.println(rs.getString("address"));  
    }  
    conn.close();  
}
```

Try:

```
getStudentAddress(307088592, "'" OR 1=1 -- ");
```

```
SELECT address FROM Student WHERE studId='307088592' AND password  
= "'" OR 1=1 -- '
```

- › Views: Can be used to provide a restricted view of a relation / relations

Professor(Id, FirstName, LastName, address)

```
CREATE VIEW Professor_public AS  
    SELECT Id, FirstName, LastName  
    FROM Professor;
```

- › SQL allows you to grant/revoke certain **privileges** to users to numerous DB objects like tables/views/stored procedures:
 - GRANT SELECT ON Professor_public TO Jimbo
 - REVOKE SELECT ON Professor_public TO Jimbo
- › Also lets you grant/revoke **privileges** (eg: *select/update/delete*) to roles
 - CREATE ROLE StudentRole
 - GRANT StudentRole TO Jimbo
 - GRANT SELECT ON Professor_public TO StudentRole

http://docs.oracle.com/database/121/SQLRF/statements_9013.htm

- › Run logic within the database server
 - Included as schema element (stored in DBMS)
 - Invoked by the application

 - › Pros:
 - Additional abstraction layer (programmers do not need to know the schema)
 - Reduced data transfer
 - Less long-held locks
 - DBMS-centric security and consistent logging/auditing (important!)

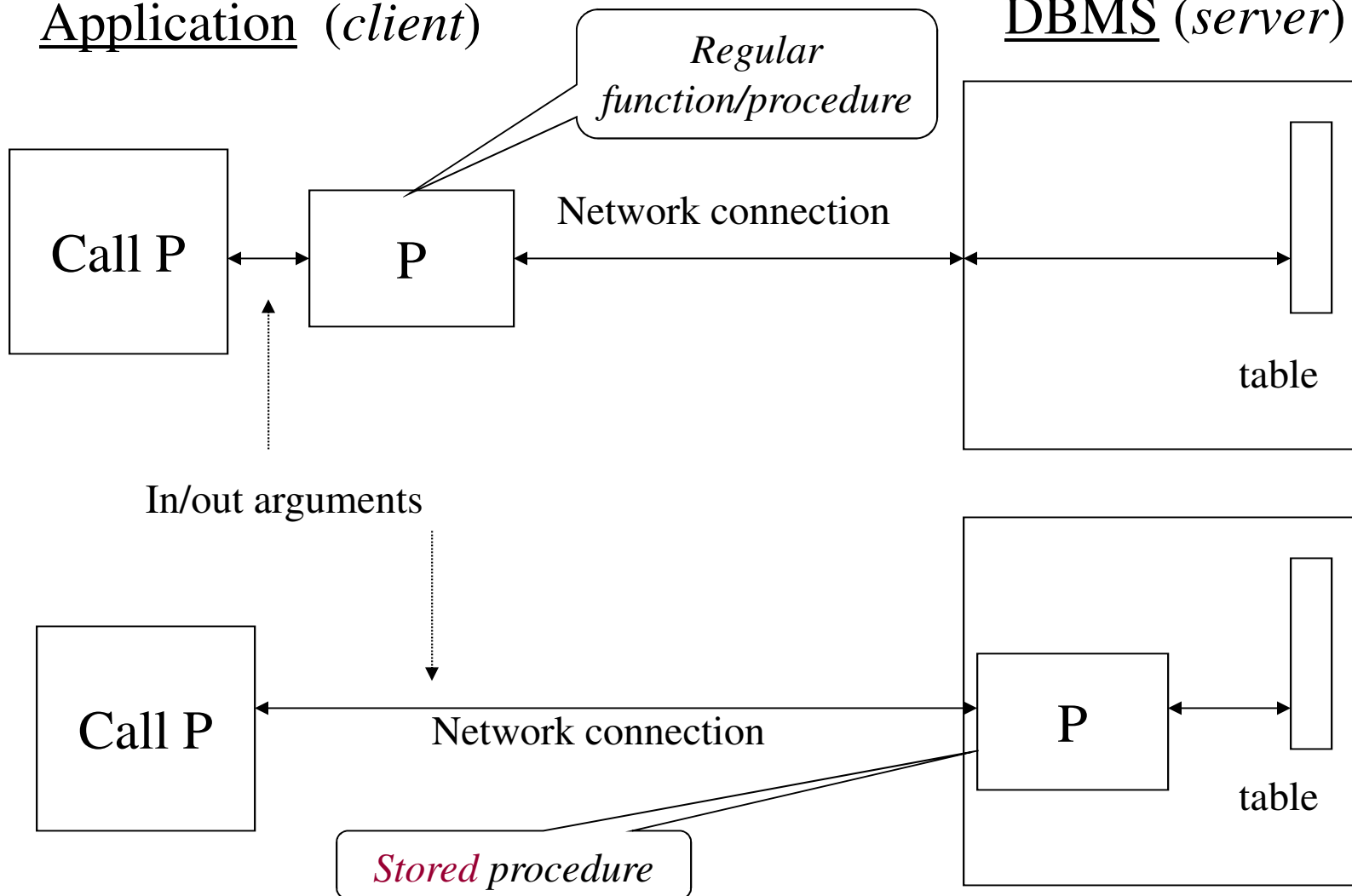
 - › Cons:
 - What if you wanted to switch DBMS?
 - rewrite all logic?
 - Language not as expressive
 - Scaling out DB tier more challenging than app server tier
-



Stored Procedures network activity

Application (*client*)

DBMS (*server*)



Stored Procedure features

- › All major database systems provide extensions of SQL to a simple, general purpose language
 - SQL:1999 Standard: SQL/PSM
 - PostgreSQL: PL/pgSQL, Oracle: PL/SQL (syntax differs!!!)
- › Procedure Declarations (with SQL/PSM)

```
CREATE PROCEDURE name ( parameter1, ..., parameterN )  
    local variable declarations  
    procedure code;
```

- › Stored Procedures can have parameters
 - of a valid SQL type (parameter types must match)
 - three different modes
 - IN arguments to procedure
 - OUT return values
 - INOUT combination of IN and OUT
 - › Stored Procedures have full access to SQL, plus extensions:
 - Local variables, loops, if-then-else conditions
-

```
create procedure RateStudent
(studId IN INTEGER, uos IN VARCHAR, result out char) AS
grade CHAR;
marks  INTEGER;
begin
SELECT SUM(mark) INTO marks
  FROM Assessment
 WHERE sid=studId AND uosCode=uos;
IF      ( marks>84 ) THEN grade := 'HD';
ELSIF   ( marks>74 ) THEN grade := 'D';
ELSIF   ( marks>64 ) THEN grade := 'CR';
ELSIF   ( marks>50 ) THEN grade := 'P';
ELSE grade := 'F';
END IF;

result := grade;
END;
```

```
create FUNCTION RateStudent_Func
(studId IN INTEGER, uos IN VARCHAR) RETURN CHAR AS
grade CHAR;
marks INTEGER;
begin
SELECT SUM(mark) INTO marks
  FROM Assessment
 WHERE sid=studId AND uosCode=uos;
IF      ( marks>84 ) THEN grade := 'HD';
ELSIF   ( marks>74 ) THEN grade := 'D';
ELSIF   ( marks>64 ) THEN grade := 'CR';
ELSIF   ( marks>50 ) THEN grade := 'P';
ELSE grade := 'F';
END IF;

RETURN grade;
END;
```

Calling Stored Procedures from JDBC

- › Use `java.sql.CallableStatement`, subclass of `Statement`
- › Calling a Stored Procedure with parameters:
(here: first IN, second an OUT parameter)

```
CallableStatement call = conn.prepareCall("{call RateStudent(?, ?, ?)}");  
call.setInt(1, 1);  
call.setString(2, "COMP5138");  
call.registerOutParameter(3, Types.VARCHAR);  
call.executeUpdate();
```

```
String result = call.getString(3);
```

- › The syntax for calling stored Functions is as follows:
 - › `CallableStatement call = conn.prepareCall("{? = call RateStudent_Func(?, ?)}");`
 - The first ? refers to the return value of the function and is also to be registered as an OUT parameter.
-

Example : Stored Procedures

- › The following function performs several queries and updates, incurring several network round trips. Rewrite as a stored procedure and change the function to call this.

```
void enrolStudent(Connection conn, int studID,  
                  String uos, String sem, int year){  
    PreparedStatement stmt = conn.prepareStatement(  
        "INSERT INTO Transcript VALUES (?, ?, ?, ?, null)");  
    stmt.setInt(1, studID);  
    stmt.setString(2, uos);  
    stmt.setString(3, sem);  
    stmt.setInt(4, year);  
    stmt.executeUpdate();  
    stmt.close();  
  
    stmt = conn.prepareStatement(  
        "UPDATE UoSOffering SET enrollment=enrollment+1"  
        + " WHERE UoSCode=? AND Semester=? AND Year=?");  
    stmt.setString(1, uos);  
    stmt.setString(2, sem);  
    stmt.setInt(3, year);  
    stmt.executeUpdate();  
    stmt.close();  
}
```

```
CREATE OR REPLACE
PROCEDURE ENROLSTUDENT (
    sid IN INTEGER,
        uos IN VARCHAR,
    sem IN VARCHAR,
    yr IN INTEGER) AS
BEGIN
    INSERT INTO Transcript VALUES (sid, uos, sem, yr, null);
    UPDATE UoSOffering SET enrollment=enrollment+1
    WHERE uoSCode=uos AND semester=sem AND year=yr;
END;
```

```
void enrolStudent(Connection conn, int studID,
                    String uos, String sem, int year) {
    CallableStatement stmt = conn.prepareCall(
        "{call ENROLSTUDENT (?, ?, ?, ?)}");
    stmt.setInt(1, studID);
    stmt.setString(2, uos);
    stmt.setString(3, sem);
    stmt.setInt(4, year);
    stmt.executeUpdate();
    stmt.close();
}
```

- › Ramakrishnan/Gehrke (3rd edition - the 'Cow' book)
 - Chapter 6; 7.5
- › Kifer/Bernstein/Lewis (2nd edition)
 - Chapter 8
- › Ullman/Widom (3rd edition of 'First Course in Database Systems')
 - Chapter 9 (*covers Stored Procedures, ESQL, CLI, JDBC and PHP*)

Research Papers and Presentations:

- › E.M. Fayo, 2005: "Advanced SQL Injection in Oracle Databases"
 - www.blackhat.com/presentations/bh-usa-05/bh-us-05-fayo.pdf

Database Documentation:

- Oracle Database Concepts
 - Java JDBC reference
 - <http://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html>
 - Java JDBC tutorials
 - <http://docs.oracle.com/javase/tutorial/jdbc/index.html>
 - **Oracle® Database JDBC Developer's Guide**
 - http://docs.oracle.com/cd/E16655_01/java.121/e17657/toc.htm
-

› Transaction Management

- Transaction Concept
- Serializability
- SQL Commands to Control Transactions

› Readings:

- Ramakrishnan/Gehrke (Cow book), Chapter 16
- Kifer/Bernstein/Lewis book, Chapter 18
- Ullman/Widom, Chapter 6.6 onwards

