

Database

Applications

Prof. Dirk Dubois

Sem 3 2022

~~Fall~~

420-331-DW



Dirk Dubois

Drop-in Tuesday 8h30 - 10h00

Database design is our primary goal

- Design tables & relationships
- Creating procedures & functions
- Writing a client-side application
 - ↳ becomes much more meaningful

Oracle → MySQL → SQL Server

↑ ↓
 same company

PL/SQL is the most popular

Oracle is very weird

PDDBORA19C is our primary DB
SQL Developer → Database development

Java for client side application

Maven
Building

JAR
Building

Unit Testing

What do you need to succeed

Notebook/Pen

Network Device → Laptop for DB

Enthusiasm

Hand written notes for memory retention

Programming exercises will be given in class.

Review previous Lectures

Have the slides & support material handy

Welcome Survey on moodle

Welcome Survey

Be inclusive

Lectures & Labs

Assignments build towards the group project

SQL today \Rightarrow PL/SQL Packages PL/SQL Review \Rightarrow Client side dev

\hookrightarrow Database design \Rightarrow Designing Java Projects \Rightarrow Designing Procedure

\hookrightarrow Project \Rightarrow DB security \Rightarrow No SQL

Some assignments will be very useful.

Project is done week 12

Mid Term 15%
Final 30%

10 assignments 30%

Group Project 25%

Ponderation

Classes 3h/week
Labs " "
Homework " "

Grinch for how to spend your time

Weekly Review 1hr/week
10 Assignments 1hr/week

Studying for exams
3h midterm
4hr final
13hr Project

45hr homework in total

Project

Excel spreadsheet → Database
Excel as the poor man's database

Take spreadsheet & turn it into a database

{ Review SQL Syntax Every week is a graded assignment
Create tables
Write & execute queries

File must run from start to finish

Always needs an SQL file

Drop errors will be ignored

Reorderability - Drop conflicts Must be dropped in the right order

Assignment #1

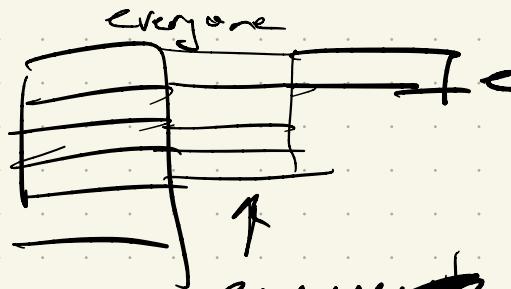
For each household show the address and #occupants

Select h.address, count(p.pid)

From household INNER JOIN person p ON
p.hid = h.id
GROUP BY h.id

2) Write a query that shows all grandfathers

Select



every parent of a parent.

everyone parent

Select p3.firstname, p3.lastname FROM

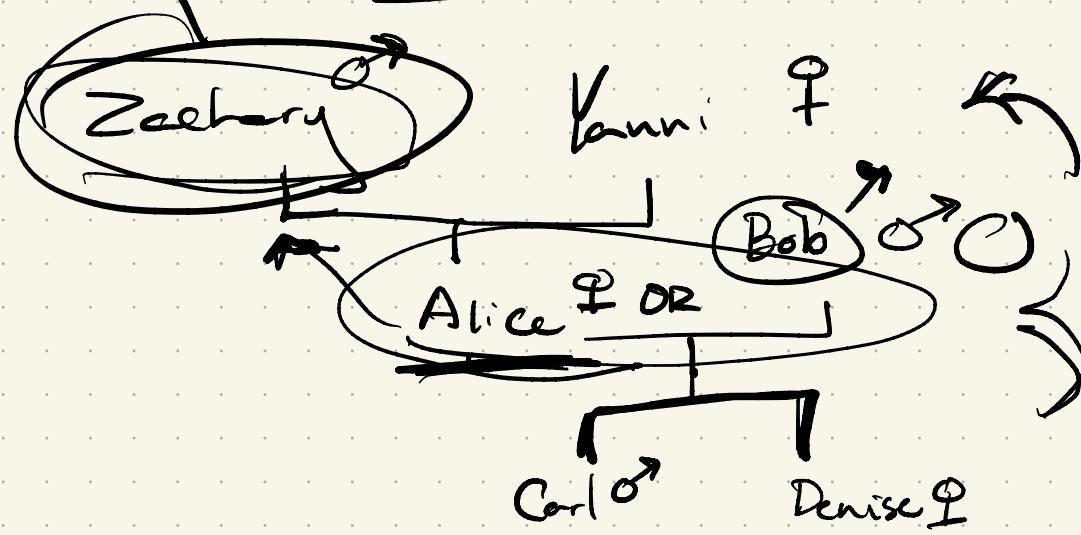
3) Select p.firstname, p.lastname

WHERE p.mother_id NOTNULL OR p.father_id = NOTNULL
AND o.id = 0001

— Need to include occupation



Select Max(Sum(~~con~~))



bob bob Zack

$$\begin{aligned} g.pid &= f.father_id \\ g.pid &= f.mother_id \end{aligned} \quad \left. \right\}$$

f.pid

Ask for slides

SQL Review

Aug 24th 2022

2 New things at the end

SQL interacts w/ Relational Databases

Learn more about this soon from last

↳ Data Control Language → Managing users / Permissions

Data Definition language

Data Manipulation language → Selects

DML

↳ Select chooses columns to display from a table

↳

↳

↳ Where clauses help us restrict what we select

↳ can chain multiple w/ AND / OR

↳ other functions such as LIKE "milk%" % → Wild card or

Aggregate Functions

↳ Multiple rows of data

↳ Sum, COUNT, MAX, MIN

↳ When we combine a single column with an aggregate function we need a group-by clause

Ex. SQL Review. SQL on moe

Select * From Books

Select * From Books WHERE Cost > 20;

Select sum(Cost) From Books WHERE cost > 20;

New ERD to help

Create & Insert Data to tables

- Create constructs tables if they don't already exist
 - ↳ Needs to include data types
 - ↳ Can't create tables that already exist
- INSERT adds data to those tables
 - ↳ Any Primary Key or foreign key constraints must be met
- Update modifies existing data
 - ↳ A where clause is needed so we know what data is to be modified

Primary / Foreign Keys

- Primary Keys ensure your rows have a unique identifier
- Foreign Keys help connect tables through their primary key
- Must DROP tables in the proper order

Code Activity #2

CREATE TABLE addresses (

 id char(4) PRIMARY KEY,
 street-number number(10) NOT NULL, ↳ Should be an varchar2(20)
 street varchar2(50) NOT NULL,
 city varchar2(50) NOT NULL,
 province varchar2(15) NOT NULL,
 postal_code char(6) NOT NULL
);

INSERT INTO addresses

GRANT modifies permissions

- GRANT can be used to modify permissions for tables, procedures & functions
- Revoke is the opposite of GRANT

SELECT, INSERT, UPDATE & more can be controlled w/ GRANT

By default all of my tables are only visible to me

GRANT Syntax

GRANT <privileges> ON <table-name> TO <username>

A2139660

A2142443

Scoping by user space

-- Shows all tables

Indices

→ provide a way to speed up read access

When designing tables, accessing table data quickly & efficiently is key

PK helps uniquely identify rows.

What if the data

CREATE INDEX
genreIndex
ON books(genre);

CREATE INDEX
costIndex

→
Index → is a side table that is used to

Genre is a fix list of categories

Cost does not have a lot of repeats

Indices are more useful for fixed list w/ repetition

Indices introduce overhead

→ Not every column should be indexed

→

Indices require maintenance

by the DBMS

→ Drop unused indices that are not being used.

First time you run an index it has to populate the index, so it is not faster yet. So it gets faster on the subsequent queries.

VIEWS

Views let us hold complex expressions

- A view is a DB object that can hold a relation defined by a query
- Views are created with a select statement and are evaluated every time they are accessed
- Use select to access the view

Allows us to leave our data nicely in the back, but the client

Dynamic VIEW

- Created w/ simple create view statement
- . Query is run everytime the view is called
- Can be complex

Materialized View

- Created w/ CREATE MATERIALIZED VIEW
- . Result of the query is cached (stored)

Insert, updates, must be maintained.

SELECTS will be faster

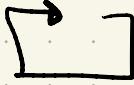
Code ex

```
CREATE VIEW title_author_view AS
SELECT b.title, b.author, b.cost
FROM books b INNER JOIN bookauthor ba
ON ba.isbn = b.isbn
INNER JOIN author a ON a.authorid = ba.author_id;
```

Packages

→ Procedural Language

PL/SQL provides a programming language to SQL



Loops



Objects

procedures
functions

invented 1993

Packages help organize our code

DB objects
are global
to our user

Naming
conflicts can
arise.

Packages encapsulate
code.

Important how we organize our code.

Can't reuse two functions of the same name since they
are global.

Packages can be shared

Packages can be seen as a complete application

→ Code can be centralized inside a package

→ Procedures and functions can be refactored inside the package

→ Types, cursors, variables and exceptions can be declared

inside a package and used across functions & procedures

Just like all our source files in Java make a complete application

Packages can have more than just procedures & functions

Appear in the packages part of the tree

Packages have two components

Package Specification

What we need to define what a user is allowed to interact with

Package Body

Packages have a public name

Specifications define procedures, functions, exceptions.
Anything needed by a user

All packages have a specification

CREATE PACKAGE <packagename> AS
OR REPLACE <packagebody>;

END <packagename>

ex.

CREATE OR REPLACE PACKAGE emp_mgmt AS
FUNCTION hire (last_name VARCHAR2
RETURN NUMBER;

PROCEDURE fire (proname& all parameters)

END emp_mgmt;

I think it's like ext

These are all public

Variables, exceptions, type can be used directly

- Since their values are publicly accessible variables, types and exceptions can be accessed without additional code

Procedures & Functions are implemented in the package body.

- Package body defines the implementation of procedures & functions
- It must contain all the procedures and functions in the package specification
- It contains additional methods that are not in the specification

CREATE OR REPLACE PACKAGE BODY <name> AS

...
Function create_dept (copy header from spec)

fill the body w/ code.

How to define a function

CREATE OR REPLACE Function get_dept_name (dept_id number)

RETURN VARCHAR2 IS
dept_name varchar2(200);
BEGIN SELECT dept_name INTO dept_name from hr.departments WHERE department_id = dept_id;

RETURN (dept_name);

END;

Variable / parameters should be meaningful BUT

NOT the same name as your table field

Recall Anonymous blocks

Call functions in anonymous blocks

DECLARE

result varchar2(200)

BEGIN

result := get_dept_id(10);
dbms_output.put_line(result);

END;

Aug 31st
Started w/ Code activity

Changing package specifications impacts the user.

Dev Codes
Package \Rightarrow Spec
indicates what is public \Rightarrow different dev calls package
subprograms



If we change the Spec, we break what new developers using our code

Google Maps API

Backwards compatibility could be critical
for existing code

Code needs to be changed

Needs to be recompiled.

↳ Code changes

↳ recompile

↳ Ship

Now code needs to be retested.

Fixing Backward Compatibility

- ↳ Don't rename
- ↳ Wrap initial function w/ a function w/ a better name.
- ↳ Warn users that original name is deprecated and they need to change their function names.

→ give clients users time

git commit -m "Meaningful to OTHER programmers"
Should be something that isn't obvious from
the git diff.
Why did you commit?

PL/SQL Subprograms: Procedures & Functions

↳ lets us group common functionality

Procedures

No Return Value

Should modify the DB
UPDATE, INSERT, DELETE

Can use other stored
procedures & functions

IN, OUT IN OUT

Can be called by Java
(JDBC)

Can be executed EXECUTE myproc();

Anon Block

Functions

Has a return value
Used to compute a result
Does Not modify the DB

Cannot execute other stored
procedures, but can use stored
functions
Can be used in select

Procedure Params have 3 Modes

IN → Passes value to the subprogram
↳ Constant

Create Procedure find min(X IN number, y IN number) {
-- code block

Can read x & y but cannot manipulate them

OUT

↳ Return a value to the caller
↳ Acts like a variable
↳ No initial value
↳ Users need to create a variable of the right type

(x IN number, y IN number, z OUT NUMBER)

z := n

DECLARE

a number := 5; b number := 4; c number;

findMin(a, b, c),
out(c);

Swap (int a, int b) Functions can only return one thing

Swap(a in __, b in __, c OUT __, d OUT __)

IN OUT

- ↳ Both passes \$ returns a value to the calling program
- ↳ initialized with the value w/ the calling context
- ↳ Acts like a variable

Look at example on slide 7

- Open our existing HRPackages.sql file
- Add a new procedure called add_dept_manager(dept_name, manager_id, dept_id) where the procedure updates the manager_id of the provided department in the HR.departments table
 - The procedure should return to the caller the id of the department that was changed
- Call the add_dept_manager procedure from an anonymous PL/SQL block and print the dept_id. Add the manager id 200 to "Treasury"



DBLab 3

Slow-purchasers

cursor for books table

or books%RowType's

~~contains~~ customer-id - array

was book-purchasers

for i in 1 .. book-purchasers.COUNT ~~Loop through~~
^(row) very array

for arr in (Select * from Books)

Loop

--Build the output string here

output := arr.isbn || ' ' || arr.name

customers := book-purchasers(arr.isbn);

for i in 1 .. customersCount

Loop

firstLastName := (Select fname, lname FROM cust

Database II

PL/SQL Varrays & cursors

Cursors & Varrays help us deal with multiple rows of data

Loops in PL/SQL

Varrays → PL/SQL Lists.

Cursors → lets us loop through queries w/ multiple data

while Loop

a := 0;

PL/SQL starts at 1,
not 0

WHILE a < 5 LOOP

a := a + 1;

dbms_output.put_line(a);

END LOOP;

for loops

FOR a in 1 .. 5 LOOP

dbms_output.put_line('a is: ' || a)

END LOOP;

add_book

- ↳ take any parameter
- ↳ if the book isbn does not exist, create the book.
- ↳ if the book exist, update only the different fields.

VARRAY → Store multiple pieces of data

- Declared as a type
- Stores information of flat type
- Resizes to fit the content (like a List in java)
- index is 1-based (not like java which is 0 based)

CREATE TYPE numberarray IS VARRAY(100) OF NUMBER;

DECLARE

myarray numberarray;

Define the ~~TYPE~~ in the Spec

BEGIN

^{Round brackets.}

 myarray(1) := 10;

 myarray(2) := 20;

 myarray(2) := 30;

END;

BULK COLLECT INTO

Works similarly to SELECT INTO.

- Requires a query that returns multiple rows
- Has to be the same type as the array is declared to hold
- Has to be able to hold all the data.

CREATE isbnarraytype IS NARRAY(100) OF VARCHAR2(200);

DECLARE

 isbnarray isbnarraytype;

Begin

 SELECT isbn BULK

Loops iterate through the Varray

- Any loop can be used
- COUNT, LIMIT, FIRST, LAST, NEXT(), PRIOR()
are all functions on the VARRAY that can help us inside the loop
- Count starts at 1

Cursors

- ↳ Way of managing a given query
- ↳ Represents multiple pieces of data
- Can be iterated over
- Used to manipulate results from a select statement that returns multiple rows
- ↳ Can be declared as types or used directly in a for loop.

BEGIN

FOR a-row IN (SELECT title FROM books)

LOOP

dbms_output.put_line(a-row.title);

END LOOP;

/

BEGIN

FOR a-row IN (SELECT * FROM books)

LOOP

dbms_output.put_line(a-row.isbn);

dbms_output.put_line(a-row.cost);

END LOOP;

END;

Cursor variable automatically contains the selected column names

With select we can control the name of the variables
↳ Members can be accessed in . notation

Responding To runtime errors & exceptions

Exceptions are a mechanism for handling errors

Applications generate exceptions when unwanted behaviours occur

- ↳ let's us communicate errors to a user
- ↳ we can define our own exceptions

System Defined Error

- NO_DATA_FOUND
- VALUE_ERROR
- DIVIDE_BY_ZERO

USER DEFINED

- ↳ Any errors

PL/SQL uses Exception Blocks to catch Errors

Exception is a block like Declare or Begin

Combined w/ WHEN keyword exceptions can be caught

Unnamed exceptions can be caught w/ 'OTHERS'

DECLARE

BEGIN

EXCEPTION

WHEN exception THEN

WHEN OTHER THEN

stuff for all exceptions, this is not great.

DECLARE

```
vname VARCHAR2(50) := 'Bob';  
vSalary NUMBER(6,2);
```

BEGIN

```
    SELECT salary INTO vSalary WHERE name = vname;  
    dbms_output.putline(vname);
```

END;

if Bob doesn't exist \Rightarrow No_data_found

vSalary is too small \Rightarrow VALUE_ERROR

More than one piece of data \Rightarrow TOO_MANY_ROWS

DECLARE

```
vname VARCHAR2(50) := 'Bob';  
vSalary NUMBER(6,2);
```

BEGIN

```
    SELECT salary INTO vSalary WHERE name = vname;  
    dbms_output.putline(vname);
```

EXCEPTION

WHEN no_data_found THEN

```
END;
```

<stuff>

DUP_VAL_ON_INDEX

RAISING CUSTOM EXCEPTIONS

Allows us to communicate our own errors to users

Better control over errors and code flow

Exceptions can be defined inside packages like any other type

These exceptions can be raised or caught

adding to
a package my-exception EXCEPTION

Raise is how we 'throw' exceptions in PL/SQL

Raise raises the exception

Exceptions bubble up until someone catches it w/ an exception block



Custom Exceptions Must be defined before they can be caught or thrown

Define an exception in a package



Raise an exception in a function
or procedure



Real World

Catches are meant help the developer.

CREATE PROCEDURE

< Update statement >

checks if it updated rows

IF SQL%NOTFOUND THEN

RAISE my_package.e_invalid_dept_id;

END IF;

- ① Define the error in specs.
- ② Give an error if no city
add exception to the get-city-one()
block

Assignment Clarification →

When there are multiple exceptions in a procedural function

As soon as you hit an exception, you move straight to the
Exception

We can have multiple begin ends in an anonymous block

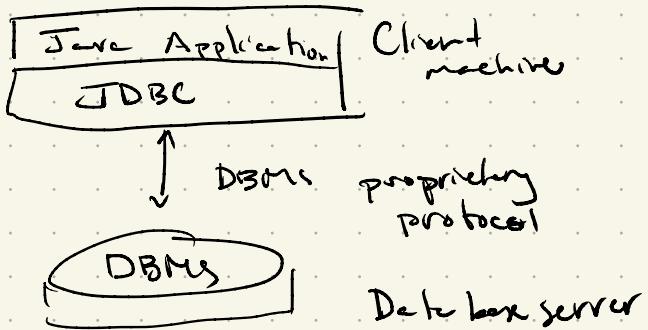
Introduction to JDBC

Dirk went very fast this lecture,
these notes are super incomplete.
Check the slides.

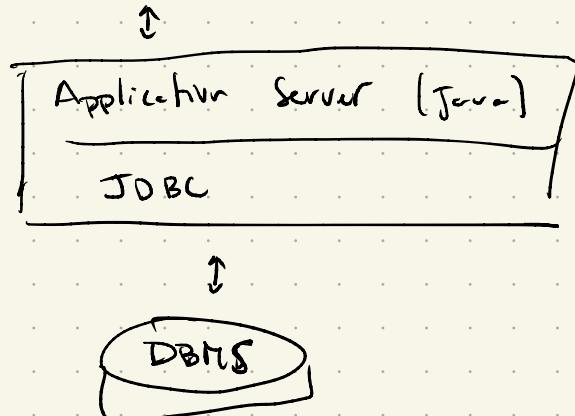
Java Data Base Connectivity lets Java talk to RDBMS

→ Java can't talk to RDBMS on its own

A library is needed to provide that functionality



Java Applet or
HTML Browser



JDBC lets us talk to multiple database technology

Each DBMS is different

- JDBC just tells us what it looks like to connect to a DB using java
- We need a driver which tells us how to connect

Types of JDBC driver

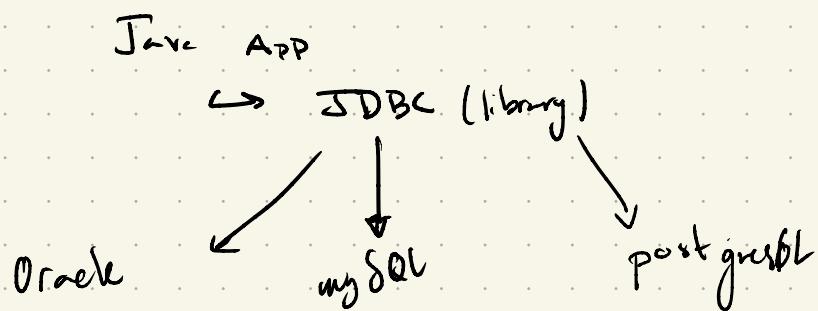
Type 1: Drivers that implements the JDBC API as mapping to another data access API.

→ Type 2: Drivers that are written

3: Drivers that are pure Java

4: Pure java and implement the network protocol
(us) → for a specific data source.

Client connects directly to the data source.



Use Maven to handle JDBC

Create a Maven project like in 3C0

Modify pom.xml to include JDBC

- This tells Maven we need ojdbc

```
<!--  
https://mvnrepository.com/artifact/com.oracle.database.jdbc/ojdbc10 -->  
<dependency>  
  <groupId>com.oracle.database.jdbc</groupId>  
  <artifactId>ojdbc10</artifactId>  
  <version>19.16.0.0</version>  
</dependency>
```

JDBC implements the java.sql interface import java.sql.*;

- ↳ Java defines a set of methods to talk to RDBMS
 - ↳ defined into the java.sql package

- Import java.sql
- Connect to DB
- Create a statement of some kind

Connecting needs a username & password

String url =

Connection conn = DriverManager.getConnection(url, User.username,
User.password)

Prepared statements send info to the database.

PreparedStatement statement = conn.prepareStatement("select * From books");
result = statement.executeQuery();

statement.setString(1, <arg>);

prepared statements are very flexible

executeQuery (query)

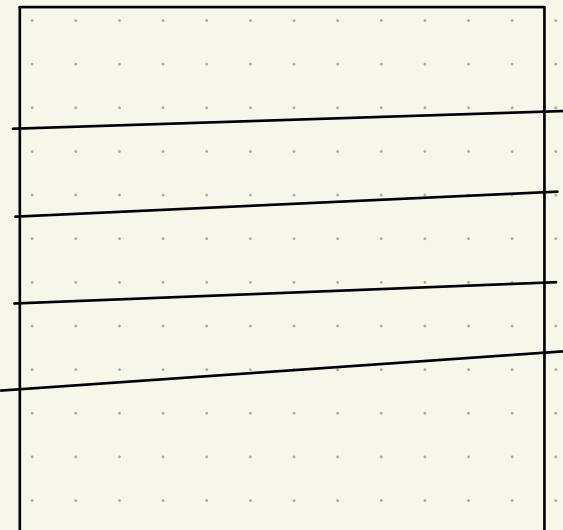
executeUpdate (query)

↳ Non-select queries (update, delete)

Result Sets are returned by queries

- What is returned behaves like a cursor
- Each result needs to be iterated through
- At a moment, a result represents a single row

← Starts here



ResultSet result =

```
Statement executeQuery( );  
returns true  
if there is another  
row, then move  
to it.  
System.
```

result.next() dynamically

each call to next() fetches a new row

```
while (result.next()) {
```

getXXX methods help fetch data

from a given column

↳ User needs to know the
Type

```
String title =  
result.getString('title');
```

```
double cost =  
result.getDouble('cost');
```

Managing Connections is important.

↳ we have to open & close connections

- Prepared Statements & Connets are all talking to the database.
- have to close Prepared Statements / Connections

```
if (!statement.isClosed()) {
```

```
    statement.close();
```

```
}
```

```
if (!conn.isClosed()) {
```

```
    conn.close();
```

```
}
```

