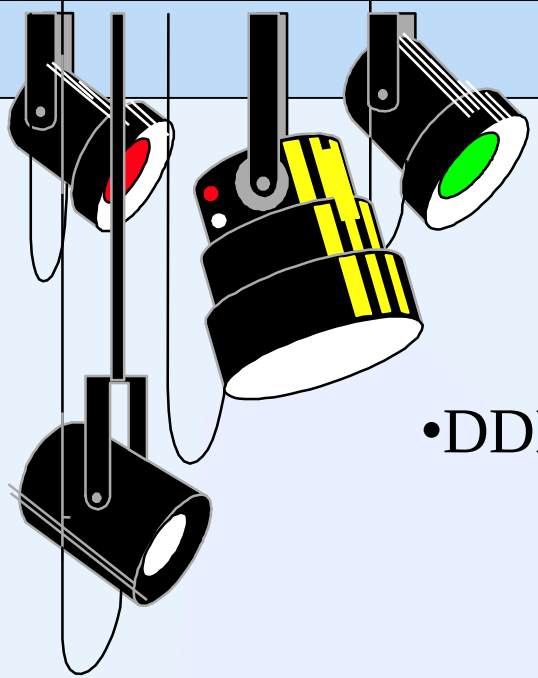
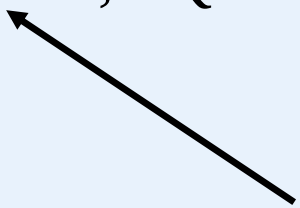


Outline: SQL and JDBC

- DDL
  - creating schemas
  - modifying schemas
- DML
  - select-from-where clause
  - group by, having, order by
  - update
  - view
- JDBC – Java Database Connectivity

Structured Query Language

- declarative or non-procedural
- DDL for data definition
- DML for query, update, view
- facility for security, integrity constraints, transactions, embedding into other 3GLs such as Cobol, C, …
- SQL89, SQL92, SQL2000?



Also referred to as **SQL2**

DDL - creating schemas

- **Create schema** *schemaname* **authorization** *user*;
- **Create table** *tablename* …
  - attributes, data types
  - constraints:
    - primary keys
    - foreign keys
      - on delete set null|cascade|set default
      - on update set null|cascade|set default
      - on insert set null|cascade|set default
    - uniqueness for secondary keys
- **Create domain** *domainname* …

DDL - Examples:

•**Create schema:**

Create schema COMPANY **authorization** JSMITH;

•**Create table:**

Create table EMPLOYEE

```
(FNAME      VARCHAR(15)     NOT NULL,
 MINIT      CHAR,
 LNAME      VARCHAR(15)     NOT NULL,
 SSN        CHAR(9)         NOT NULL,
 BDATE      DATE,
 ADDRESS    VARCHAR(30),
 SEX        CHAR,
 SALARY     DECIMAL(10, 2),
 SUPERSSN   CHAR(9),
 DNO        INT             NOT NULL,
```

PRIMARY KEY(SSN),

FOREIGN KEY(SUPERSSN) **REFERENCES** EMPLOYEE(SSN),

FOREIGN KEY(DNO) **REFERENCES** DEPARTMENT(DNUMBER));

DDL - Examples:

•**Specifying constraints:**

```
Create table EMPLOYEE
(…,
DNO       INT                    NOT NULL          DEFAULT 1,
CONSTRAINT EMPPK
    PRIMARY KEY(SSN),
CONSTRAINT EMPSUPERFK
FOREIGN KEY(SUPERSSN) REFERENCES EMPLOYEE(SSN)
    ON DELETE SET NULL  ON UPDATE CASCADE,
CONSTRAINT EMPDEPTFK
FOREIGN KEY(DNO) REFERENCES DEPARTMENT(DNUMBER)
     ON DELETE SET DEFAULT  ON UPDATE CASCADE);
```

•**Create domain:**

```
CREATE DOMAIN SSN_TYPE AS CHAR(9);
```

set null or cascade: strategies to maintain data consistency

Employee

ssn	… …	supervisor
123456789		234589710
… …		
234589710		null

delete

Employee

ssn	… …	supervisor
123456789		234589710
… …		
234589710		null

not reasonable

delete

cascade

delete

set null or cascade: strategies to maintain data consistency

Employee

ssn	… …	supervisor
123456789		234589710
… …		
234589710		null

delete

Employee

ssn	… …	supervisor
123456789		null
… …		
234589710		null

reasonable

set null

delete

set default: strategy to maintain data consistency

Department

DNUMBER	… …	… …
1		… …
… …		
4		… …

delete

Employee

ssn	… …	DNO
123456789		4
… …		
234589710		… …

change this value to the default value 1.

Cascade – a strategy to enforce referential integrity

Employee

ssn	…	
123456789		
…		

delete

Works_On

ssn	pno	hours
123456789	…	20
…		

Works_On

ssn	pno	hours
…		

Delete cascading.

DDL - modifying schemas

- **drop schema** *schemaname* **cascade|restrict**
- **drop table** *tablename* **cascade|restrict**
- **alter table** *tablename* **add|drop** *attributename* **cascade|restrict**
- **drop constraint** ...

DDL - Examples:

- **drop schema**

```
DROP SCHEMA CAMPANY CASCADE;
DROP SCHEMA CAMPANY RESTRICT;
```

- **drop table**

```
DROP TABLE EMPLOYEE CASCADE;
DROP TABLE EMPLOYEE RESTRICT;
```

- **alter table**

```
ALTER TABLE COMPANY.EMPLOYEE
        ADD JOB VARCHAR(12);
ALTER TABLE COMPANY.EMPLOYEE
        DROP ADDRESS CASCADE;
```

DDL - Examples:

•**alter table** (continue)

ALTER TABLE COMPANY.DEPARTMENT
ALTER MGRSSN **DROP** DEFAULT;
ALTER TABLE COMPANY.DEPARTMENT
ALTER MGRSSN **SET DEFAULT** “33344555”;

•**drop constraints**

ALTER TABLE COMPANY.EMPLOYEE
DROP CONSTRAINT EMPSUPERFK **CASCADE**;

ALTER TABLE COMPANY.EMPLOYEE
ADD CONSTRAINT (EMPSUPERFK FOREIGN KEY(SUPERSSN) REFERENCE EMPLOYEE(SSN));

DML - Queries (the Select statement)

select *attribute list*

from *table list*

where *condition*

group by *expression*

having *expression*

order by *expression ;*

Select *fname, salary* **from** *employee* ***where*** *salary > 30000* ⇨

$\pi_{\text{fname, salary}}(\sigma_{\text{salary}>30000}(\text{Employee}))$

Select *salary* **from** *employee*;

Salary
30000
40000
25000
43000
38000
25000
25000
55000

See Fig. 7.6 for the relation employee.

Duplicates are possible!

Select *fname*, *salary* **from** *employee* ***where*** *salary > 30000*;

Fname	***Salary***
Franklin	40000
Jennifer	43000
Ramesh	38000
James	55000

Select distinct *salary* **from** *employee;*

Salary
30000
40000
25000
43000
38000
55000

Select Distinct *suppresses duplicates;* ***Select All*** *does not -* ***select all*** *is the default*

Select average(*salary)* **from** *employee;*

Average(Salary)
37625

Select average(distinct *salary)* **from** *employee;*

Average(distinct Salary)
38500

Select *d.dnumber, dname, dlocation*

from *department d, dept_locations l*

where *d.dnumber=l.dnumber;*

- *aliasing*
- *basic **join** condition*
- *3 departments, 5 locations, 5 rows in the result*

d.dnumber	***dname***	***location***
1	Headquarters	Houston
4	Administration	Stafford
5	Research	Bellaire
5	Research	Sugarland
5	Research	Houston

Select *s.ssn, s.lname, r.lname*

from *employee s, employee r*

where *s.ssn=r.superssn;*

- ***Recursive join*** *- same relation - aliases required*

s.ssn	***s.lname***	***r.lname***
333445555	Wong	Smith
888665555	Borg	Wong
987654321	Wallace	Zelaya
888665555	Borg	Wallace
333445555	Wong	Nanayan
333445555	Wong	English
987654321	Wallace	Jabbar

Ordering the result set:

Select *s.ssn, s.lname, r.lname*

from *employee s, employee r*

where *s.ssn=r.superssn*

order by *s.lname, r.lname;*

s.ssn	***s.lname***	***r.lname***
888665555	Borg	Wallace
888665555	Borg	Wong
987654321	Wallace	Jabbar
987654321	Wallace	Zelaya
333445555	Wong	English
333445555	Wong	Nanayan
333445555	Wong	Smith

Summarizing underlying rows:

Select *s.ssn, s.lname, count(r.lname)*

from *employee s, employee r*

where *s.ssn=r.superssn*

group by *s.ssn, s.lname;*

s.ssn	***s.lname***	***r.lname***
888665555	Borg	2
333445555	Wong	3
987654321	Wallace	2

Eliminating Groups from the result:

Select *s.ssn, s.lname, count(r.lname)*

from *employee s, employee r*

where *s.ssn=r.superssn*

group by *s.ssn, s.lname*

having *count(r.lname) < 3;*

s.ssn	***s.lname***	***count(r.lname)***
888665555	Borg	2
987654321	Wallace	2

Use of *:

select * **from** *dept_locations;*

dnumber	***dlocation***
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

select *count(*)* **from** *dept_locations;*

Count(*)
5

select *count(*)* **from** *dept_locations*

where *dlocation='Houston';*

Count(*)
2

Use of Like and Between:

select * **from** *dept_locations*

where *dlocation like '%o%';*

dnumber	***dlocation***
1	Houston
4	Stafford
5	Houston

In Access, wildcard is represented by *, instead of %.

Select *fname, salary* **from** *employee* ***where*** *salary*

between *30000* **and** *50000;*

fname	***salary***
Franklin	40000
Jennifer	43000
Ramesh	38000

Subqueries:

select *ssn*, *fname* **from** *employee*

where *ssn* **in**

*(***select** *essn* **from** *dependent);*

ssn	***fname***
333445555	Franklin
987654321	Jennifer
123456789	John

The above is a special case of a comparison operator followed by **any/some**:

select *ssn*, *fname* **from** *employee*

where *ssn* **= any**

*(***select** *essn* **from** *dependent);*

Other possibilities: > any >= any < any <= any <> any

Subqueries:

comparison operator can be followed by **all**:

Select *ssn, fname, salary* **from** *employee* ***where*** *salary* > **all** *(**select** salary **from** employee **where** dno=4);*

ssn	***fname***	***salary***
888665555	James	55000

Note that the inner query needs only to be executed once.

Correlated Subqueries:

The inner and outer query are related.

Conceptually, the subquery is executed once for each row of the outer query:

Select *dno, ssn, fname* **from** *employee e* ***where*** *salary*

>= all *(**select** salary **from** employee x **where** x.dno=e.dno);*

dno	***ssn***	***fname***
1	888665555	James
4	987654321	Jennifer
5	333445555	Franklin

Correlated Subqueries:

The inner and outer query are related.

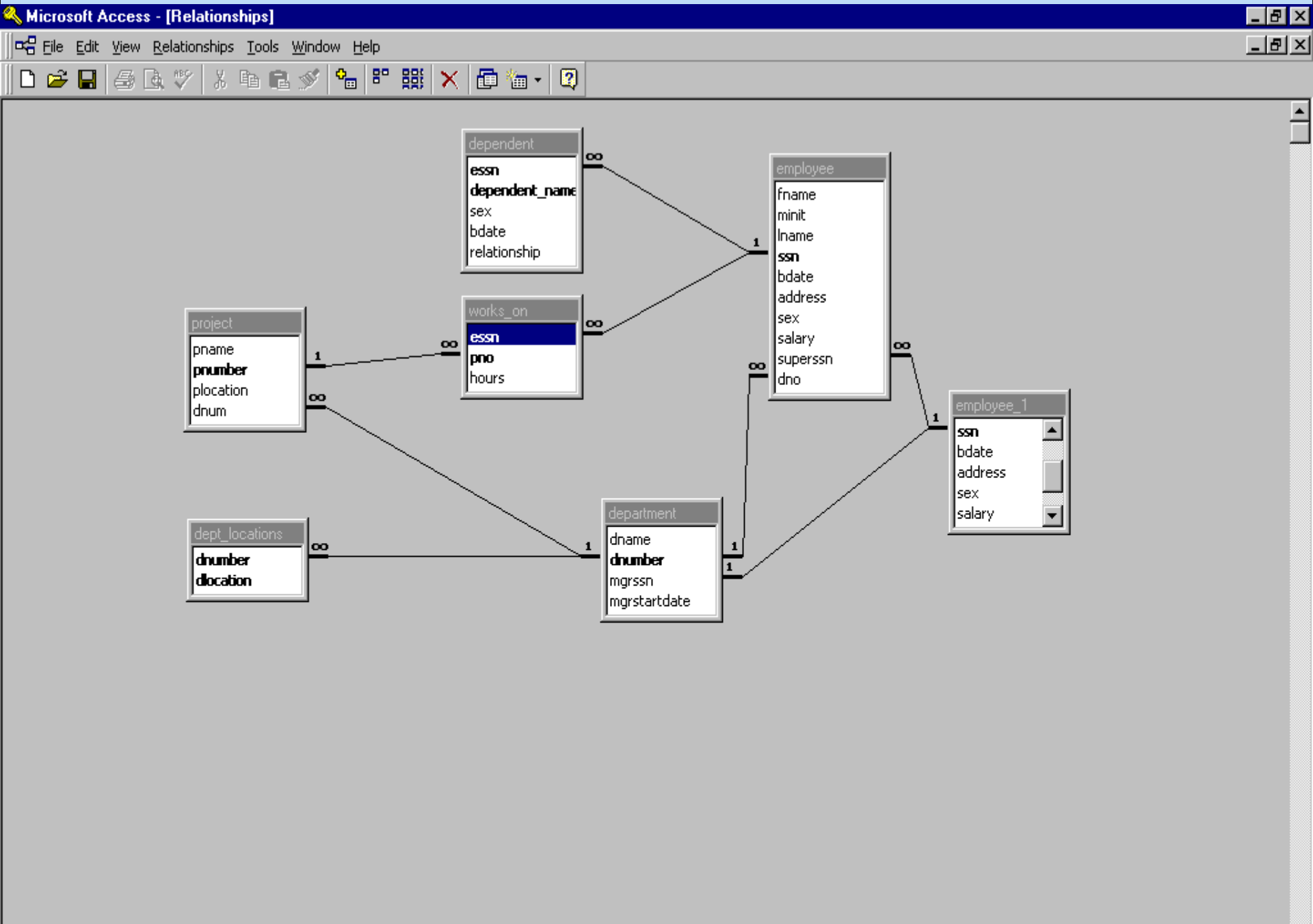
Conceptually, the subquery is executed once for each row of the outer query:

Select *dno, ssn, fname* **from** *employee e* ***where*** *salary*

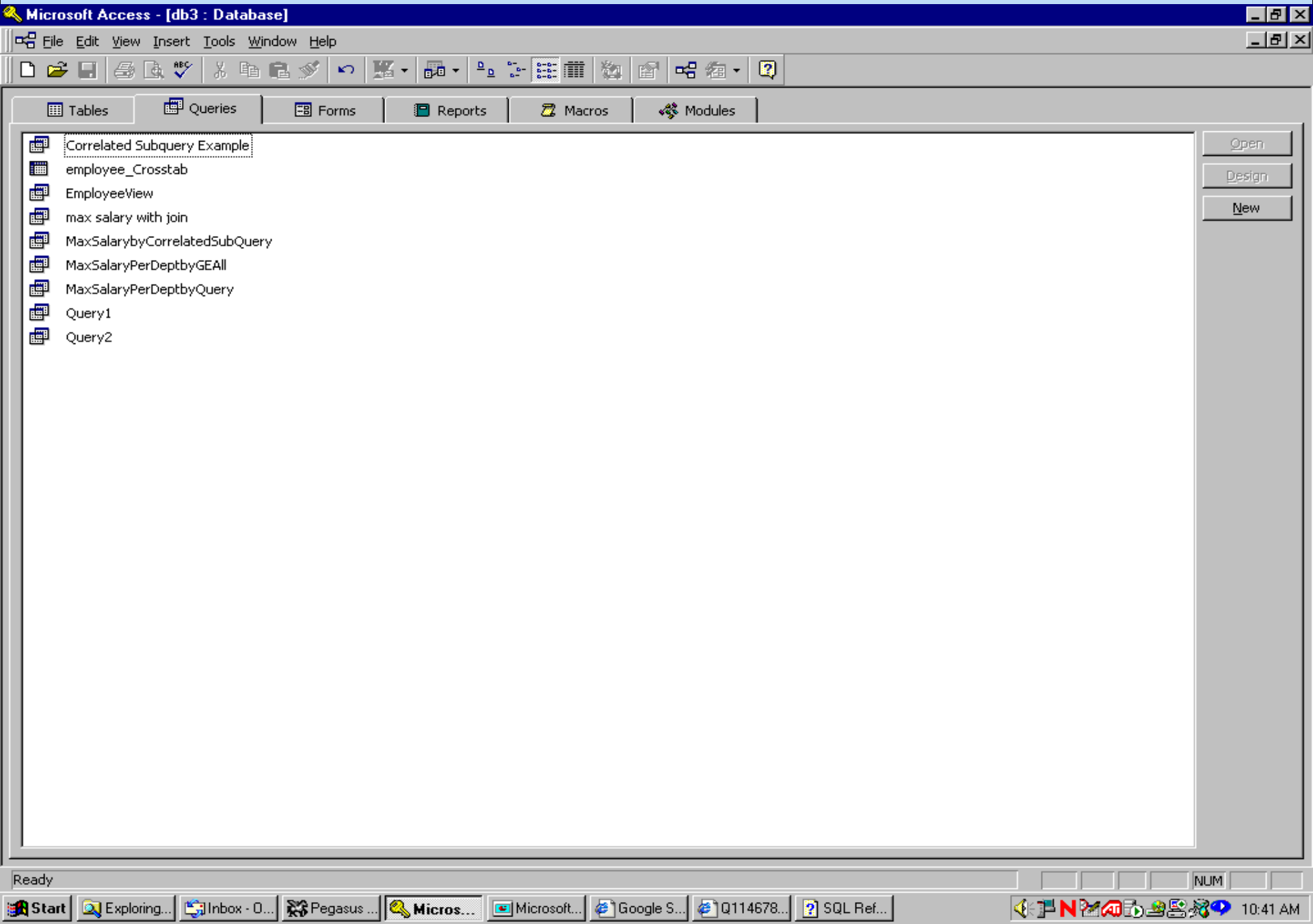
= (select max*(salary)* ***from*** *employee x* ***where*** *x.dno=e.dno);*

dno	***ssn***	***fname***
1	888665555	James
4	987654321	Jennifer
5	333445555	Franklin

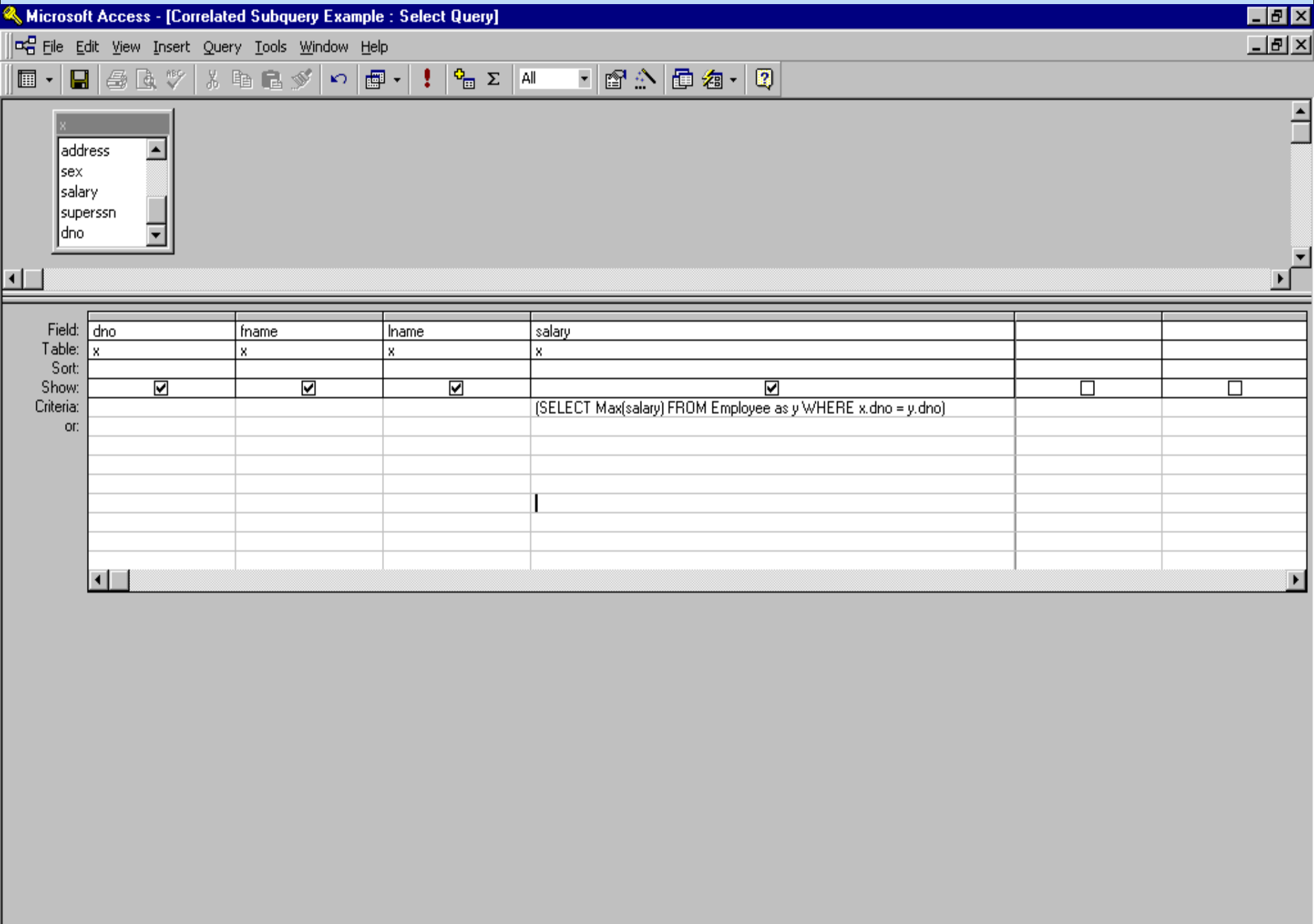
SQL


Microsoft Access - [Relationships]
File Edit View Relationships Tools Window Help
dependent
essn
dependent_name
sex
bdate
relationship
employee
fname
minit
lname
ssn
bdate
address
sex
salary
superssn
dno
works_on
essn
pno
hours
project
pname
pnumber
plocation
dnum
employee_1
ssn
bdate
address
sex
salary
department
dname
dnumber
mgrssn
mgrstartdate
dept_locations
dnumber
dlocation
1
∞

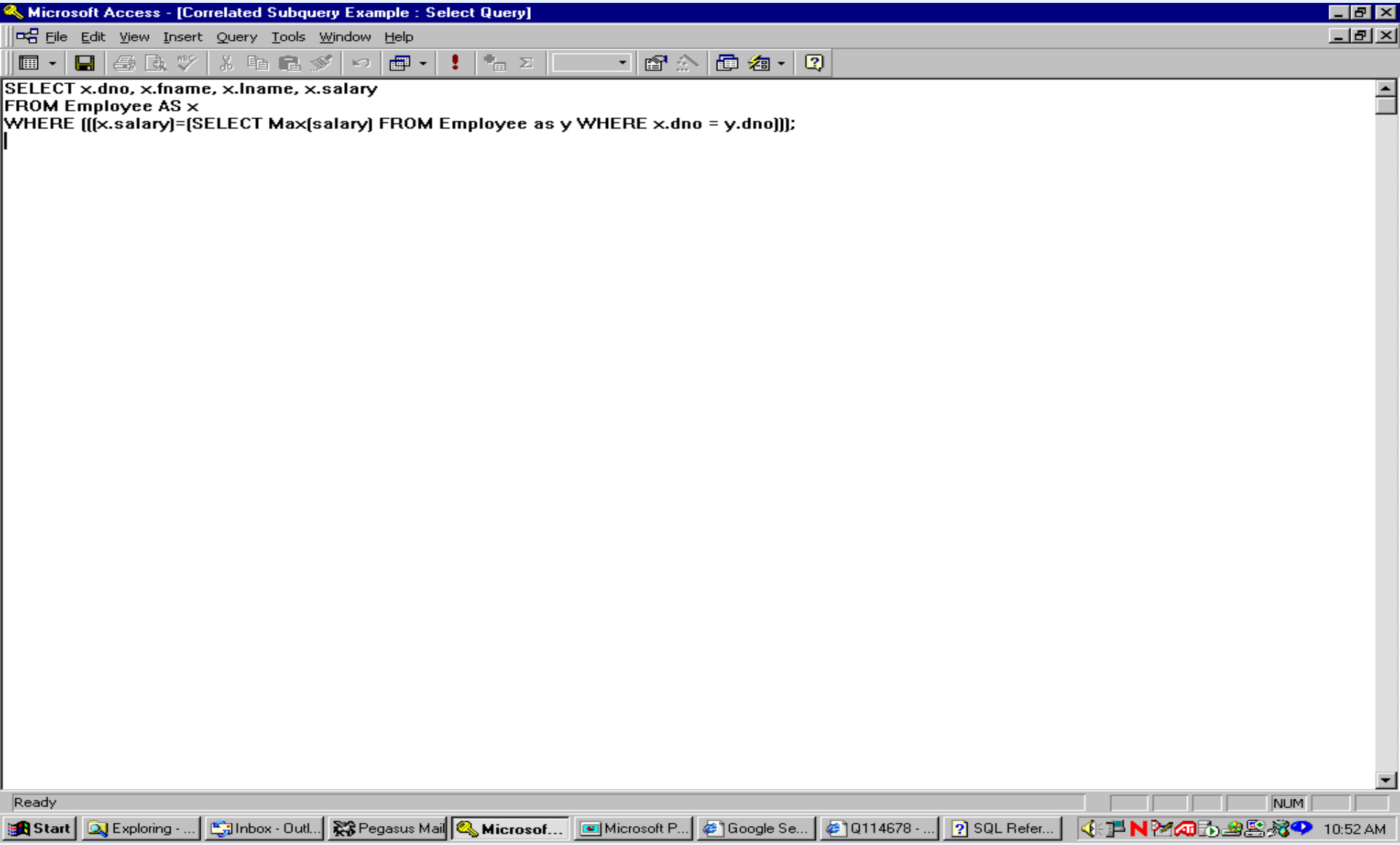
SQL


Microsoft Access - [db3 : Database]
File Edit View Insert Tools Window Help
Tables
Queries
Forms
Reports
Macros
Modules
Correlated Subquery Example
employee_Crosstab
EmployeeView
max salary with join
MaxSalarybyCorrelatedSubQuery
MaxSalaryPerDeptbyGEAll
MaxSalaryPerDeptbyQuery
Query1
Query2
Open
Design
New
Ready
NUM
Start
Exploring...
Inbox - O...
Pegasus ...
Micros...
Microsoft...
Google S...
Q114678...
SQL Ref...
10:41 AM

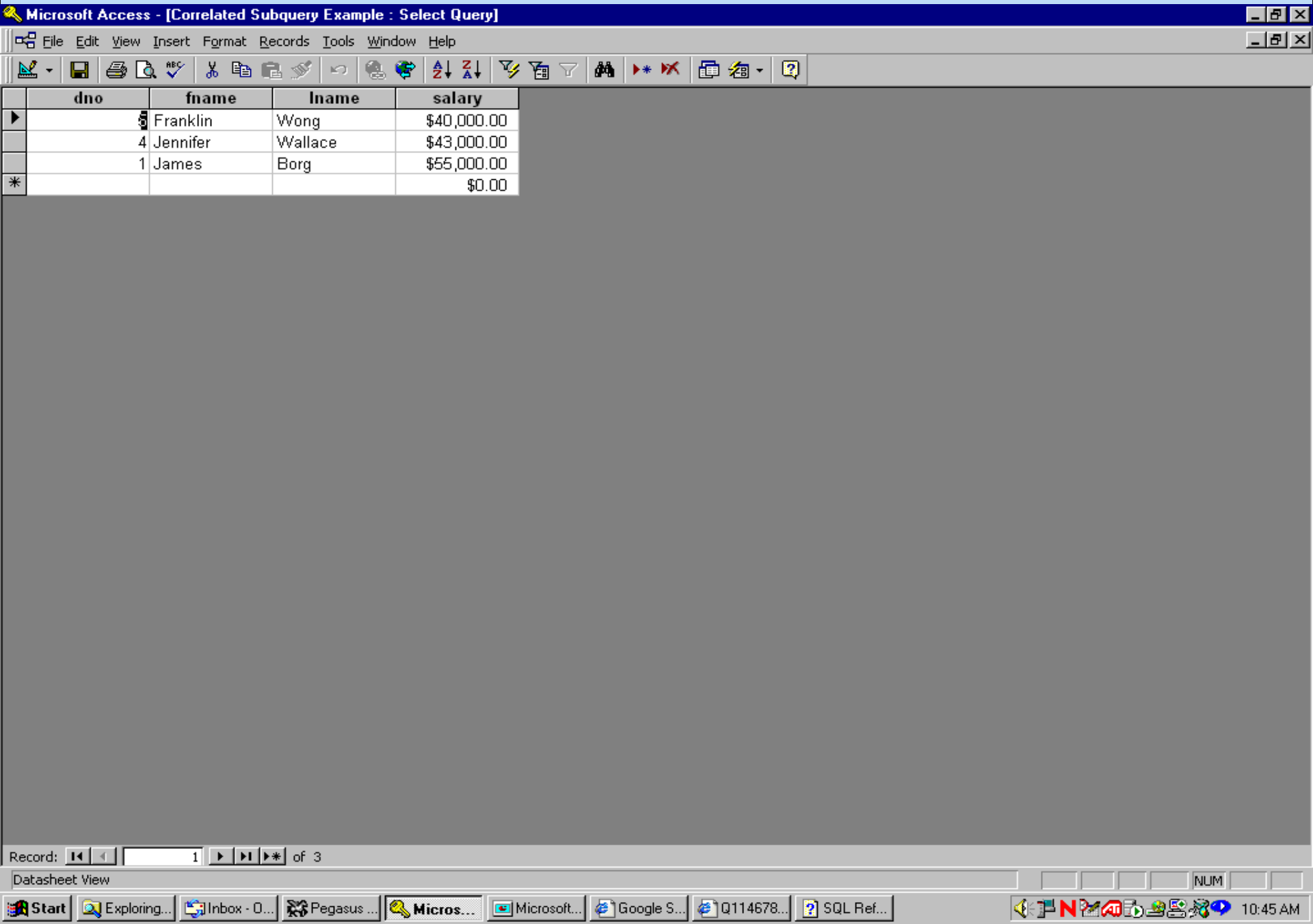
SQL


Microsoft Access - [Correlated Subquery Example : Select Query]
File Edit View Insert Query Tools Window Help
All
x
address
sex
salary
superssn
dno
Field: dno fname lname salary
Table: x x x x
Sort:
Show:
Criteria: (SELECT Max(salary) FROM Employee as y WHERE x.dno = y.dno)
or:

SQL


Microsoft Access - [Correlated Subquery Example : Select Query]
File Edit View Insert Query Tools Window Help
SELECT x.dno, x.fname, x.lname, x.salary
FROM Employee AS x
WHERE (((x.salary)=(SELECT Max(salary) FROM Employee as y WHERE x.dno = y.dno)));
Ready
NUM
Start
Exploring - ...
Inbox - Outl...
Pegasus Mail
Microsof...
Microsoft P...
Google Se...
Q114678 - ...
SQL Refer...
10:52 AM

SQL


Microsoft Access - [Correlated Subquery Example : Select Query]
File Edit View Insert Format Records Tools Window Help
dno fname lname salary
5 Franklin Wong $40,000.00
4 Jennifer Wallace $43,000.00
1 James Borg $55,000.00
$0.00
Record: 1 of 3
Datasheet View
NUM
Start
Exploring...
Inbox - O...
Pegasus ...
Micros...
Microsoft...
Google S...
Q114678...
SQL Ref...
10:45 AM

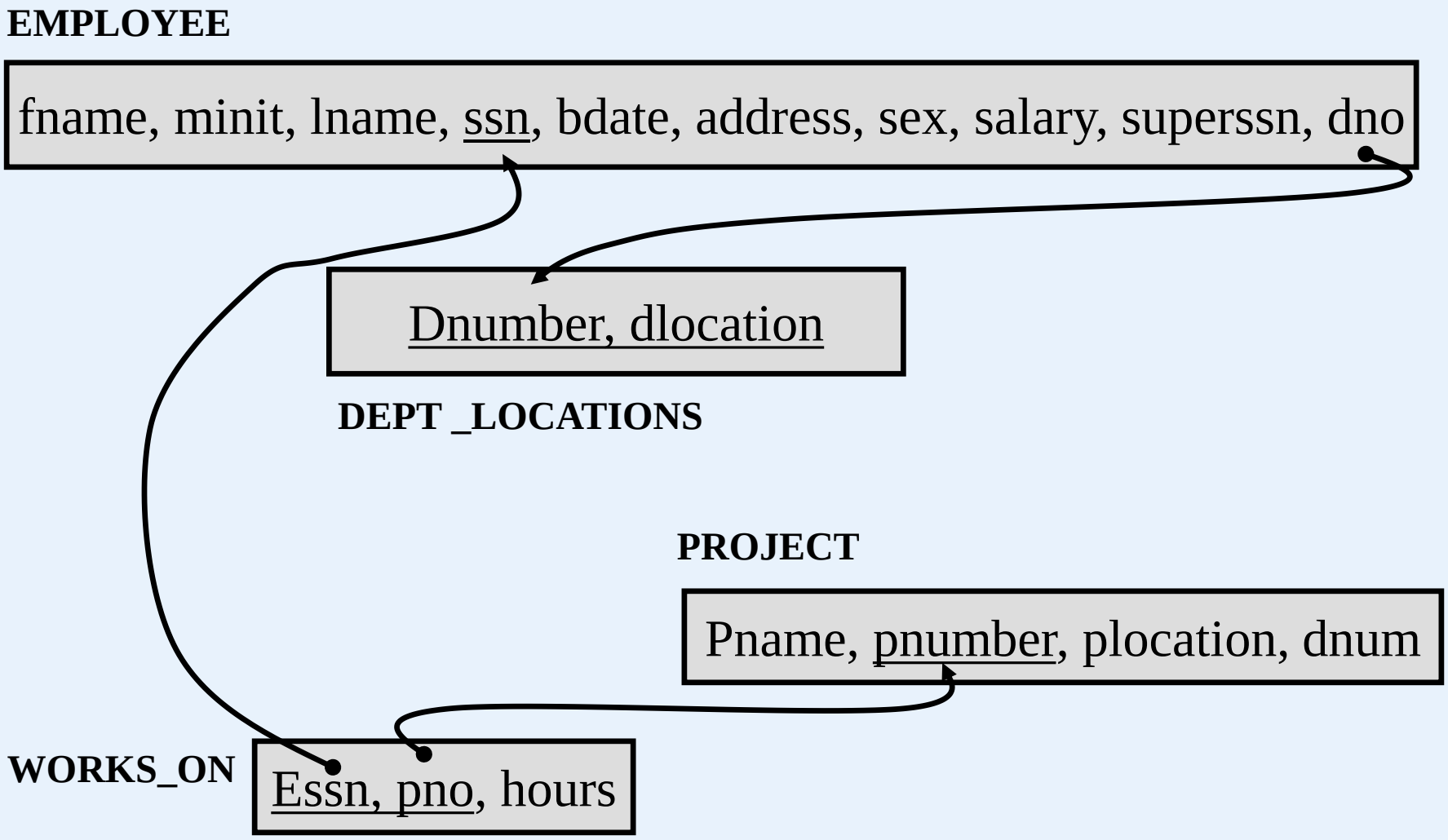
Correlated Subquery example:

Suppose we want to find out who is working on a project that is not located where their department is located.

- Note that the Project table has the location for the project
- Note that the Works_on relates employees to projects
- Note that the Employee table has the department number for an employee, and that Dept_locations has the locations for the department

We'll do this in two parts:

- a join that relates employees and projects (via works_on)
- a subquery that obtains the department locations for a given employee


EMPLOYEE
fname, minit, lname, ssn, bdate, address, sex, salary, superssn, dno
Dnumber, dlocation
DEPT _LOCATIONS
PROJECT
Pname, pnumber, plocation, dnum
WORKS_ON
Essn, pno, hours

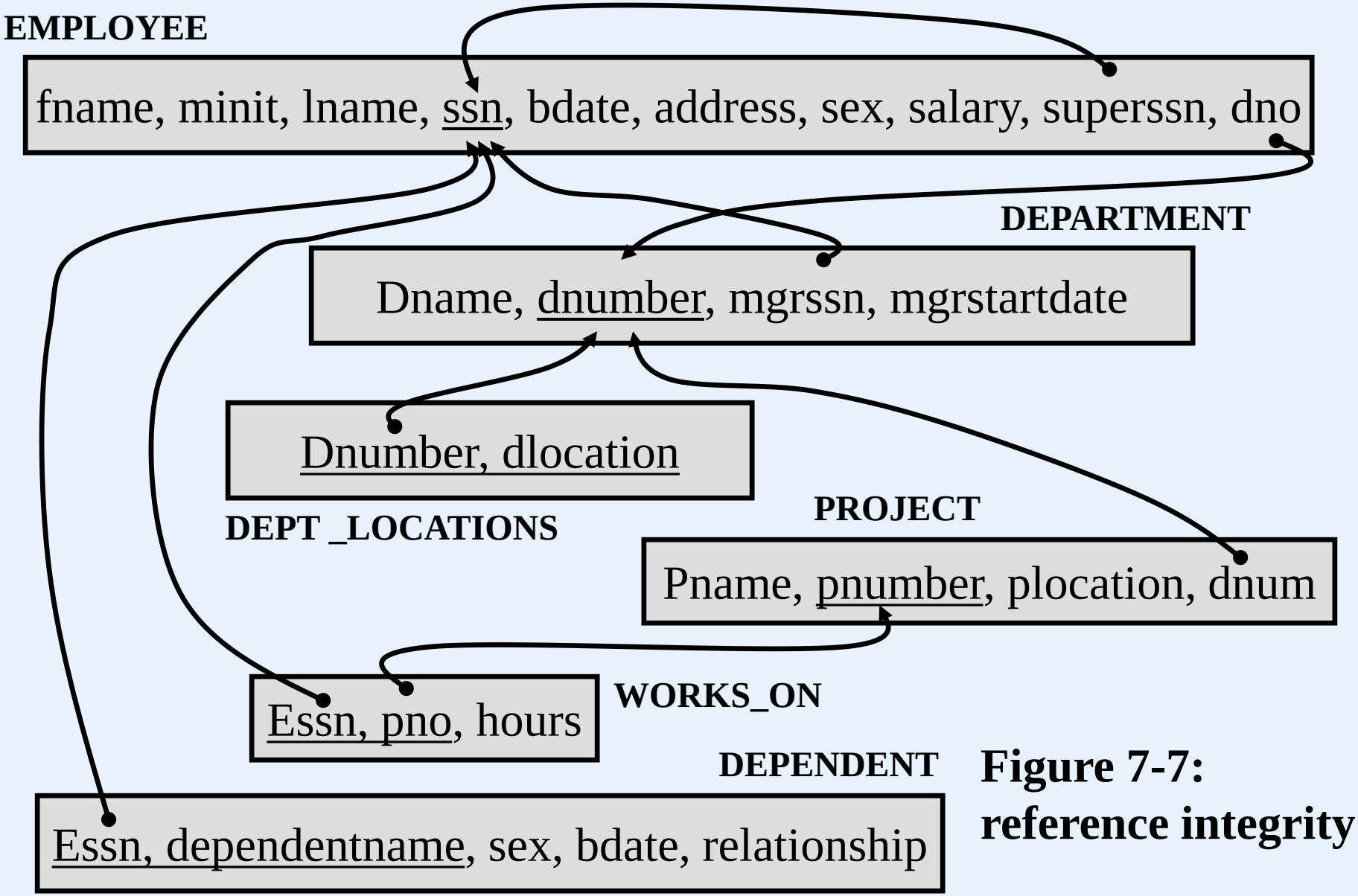


Figure 7-7: reference integrity

Correlated Subqueries:

A 3-way join to bring related employee and project data together:

```
SELECT employee.ssn, employee.fname, employee.lname,
        project.pnumber, project.plocation
FROM employee, project, works_on
WHERE
employee.ssn = works_on.essn and
project.pnumber = works_on.pno
```

A 3-way join

We'll see this join again where Inner Joins are discussed

Correlated Subqueries:

Now we incorporate a correlated subquery to restrict the result to those employees working on a project that is not where their department is located:

```
SELECT employee.ssn, employee.fname, employee.lname,
        project.pnumber, project.plocation
FROM employee, project, works_on
WHERE
employee.ssn = works_on.essn and
project.pnumber = works_on.pno and
plocation NOT IN
        (SELECT dlocation FROM dept_locations WHERE
        dnumber=employee.dno);
```

Correlated Subqueries:

Now we incorporate a correlated subquery to restrict the result to those employees working on a project that is not where their department is located:

```
SELECT employee.ssn, employee.fname, employee.lname,
        project.pnumber, project.plocation
FROM employee x, project, works_on
WHERE
employee.ssn = works_on.essn and
project.pnumber = works_on.pno and
plocation NOT IN
        (SELECT dlocation FROM dept_locations y WHERE
        y.dnumber = x.dno);
```

Subqueries with Exists and Not Exists:

Who has dependents?

```
SELECT fname, lname FROM employee
WHERE
EXISTS (SELECT * FROM dependent where essn=ssn);
```

Who does not have dependents?

```
SELECT fname, lname FROM employee
WHERE
NOT EXISTS (SELECT * FROM dependent where essn=ssn);
```

Subqueries with Exists and Not Exists:

Who is working on every project?

This is not a simple query!

```
SELECT e.ssn, e.fname, e.lname
FROM employee AS e
WHERE
NOT EXISTS
    (SELECT * FROM project AS p WHERE
    NOT EXISTS
        (SELECT * FROM works_on AS w WHERE w.essn=e.ssn
        AND w.pno=p.pno));
```

There is no project that the employee does not work on.

Example:

EMPLOYEE

ssn	fname	lname
1	… …	… …
2	… …	… …
3	… …	… …

WORK_ON

essn	PNo	hours
1	1	...
1	2	...
2	3	...
3	1	...
3	2	...
3	3	...

PROJECT

PNo	Pname	…
1	… …	…
2	… …	…
3	… …	…

For each employee *e*, check whether there is any project *p* in the result obtained by evaluating the following query.

```
SELECT * FROM project AS p WHERE
NOT EXISTS
    (SELECT * FROM works_on AS w
    WHERE w.essn=e.ssn AND w.pno=p.pno);
```

The result is a set of projects. On each of them, *e* doesn't work.

If not, *e* must be an employee who works on all projects.

Consider the employee with ssn = 1. Since there is a project with PNo = 3 in the result, he does not work on all projects.

EMPLOYEE

ssn	fname	lname
1	… …	… …
2	… …	… …
3	… …	… …

e

Project

PNo	Pname	…
1	… …	…
2	… …	…
3	… …	…

p

WORK_ON

essn	PNo	hours
1	1	…
1	2	…
2	3	…
3	1	…
3	2	…
3	3	…

w

For e = 1, PNo = 1 not in the result.

EMPLOYEE

ssn	fname	lname
1	… …	… …
2	… …	… …
3	… …	… …

e

Project

PNo	Pname	…
1	… …	…
2	… …	…
3	… …	…

p

WORK_ON

essn	PNo	hours
1	1	…
1	2	…
2	3	…
3	1	…
3	2	…
3	3	…

w

For e = 1, PNo = 2 not in the result.

EMPLOYEE

ssn	fname	lname
1	… …	… …
2	… …	… …
3	… …	… …

e

Project

PNo	Pname	…
1	… …	…
2	… …	…
3	… …	…

p

WORK_ON

essn	PNo	hours
1	1	…
1	2	…
2	3	…
3	1	…
3	2	…
3	3	…

w

For e= 1, PNo = 3 in the result.

Consider the employee with ssn = 2. Since there is two projects with Pno = 1 and PNo = 2 in the result, he does not work on all projects.

EMPLOYEE

ssn	fname	lname
1	… …	… …
2	… …	… …
3	… …	… …

e

Project

PNo	Pname	…
1	… …	…
2	… …	…
3	… …	…

p

WORK_ON

essn	PNo	hours
1	1	…
1	2	…
2	3	…
3	1	…
3	2	…
3	3	…

w

For e= 2, PNo = 1 in the result.

Consider the employee with ssn = 3. Since there is no project in the result, he work on all projects.

EMPLOYEE

ssn	fname	lname
1	… …	… …
2	… …	… …
3	… …	… …

e

Project

PNo	Pname	…
1	… …	…
2	… …	…
3	… …	…

p

WORK_ON

essn	PNo	hours
1	1	…
1	2	…
2	3	…
3	1	…
3	2	…
3	3	…

w

For e= 3, PNo = 1 not in the result.

Consider the employee with ssn = 3. Since there is no project in the result, he work on all projects.

EMPLOYEE

ssn	fname	lname
1	… …	… …
2	… …	… …
3	… …	… …

e

Project

PNo	Pname	…
1	… …	…
2	… …	…
3	… …	…

p

WORK_ON

essn	PNo	hours
1	1	…
1	2	…
2	3	…
3	1	…
3	2	…
3	3	…

w

For e= 3, PNo = 2 not in the result.

Consider the employee with ssn = 3. Since there is no project in the result, he work on all projects.

EMPLOYEE

ssn	fname	lname
1	… …	… …
2	… …	… …
3	… …	… …

e

Project

PNo	Pname	…
1	… …	…
2	… …	…
3	… …	…

p

WORK_ON

essn	PNo	hours
1	1	...
1	2	...
2	3	...
3	1	...
3	2	...
3	3	...

w

For e = 3, PNo = 3 not in the result.

Renaming the result set:

```
SELECT fname AS FirstName, lname AS Surname
FROM employee;
```

Aggregate functions:

AVG, SUM, COUNT, MAX, MIN

```
Select count(*)
from employee
```

Number of employees - count number of rows

```
Select count(superssn)
from employee
```

Number of employees who have supervisors - count ignores nulls

```
Select count(distinct superssn)
from employee
```

*Number of employees who are supervisors - **doesn't work in Access!***

Aggregate functions are normally used with Group By clause:

Select *s.ssn, s.lname, count(r.lname)*

from *employee s, employee r*

where *s.ssn=r.superssn*

group by *s.ssn, s.lname;*

s.ssn	***s.lname***	***r.lname***
888665555	Borg	2
333445555	Wong	3
987654321	Wallace	2

Nulls:

Some fields are designed so that a value is not required

In figure 7.1(a) on page 189, some fields have NOT NULL specified - these must be assigned a value on INSERT. Others do not have this specification - the default is Null, unless you override that

Specific constructs are used to test for nulls:

```
select fname, lname
from employee
where superssn is null;
```

Who does not have a supervisor?

Special joins:

Outer join table

this example is a right outer join - lists every department regardless of whether or not it has a manager

```
SELECT department.dnumber, department.dname,
employee.fname, employee.lname
FROM
employee RIGHT OUTER JOIN department
ON employee.ssn = department.mgrssn;
```

Special joins:

Inner join table

this example is an inner join - lists employees and their departments

```
SELECT department.dnumber, department.dname, employee.fname,
employee.lname
FROM
department INNER JOIN employee
ON department.dnumber = employee.dno;

SELECT department.dnumber, department.dname, employee.fname,
employee.lname
FROM department, employee
WHERE department.dnumber = employee.dno;
```

Special joins:

Inner join table with a Where clause

this example is an inner join - lists employees and their departments, but only for the Research department

```
SELECT department.dnumber, department.dname,
employee.fname, employee.lname
FROM
department INNER JOIN employee
ON department.dnumber = employee.dno
WHERE dname = 'Research';
```

Special joins:

Inner join table with a Where clause

this example is an inner join - lists employees and their departments, but only for the Research department

```
SELECT department.dnumber, department.dname,
employee.fname, employee.lname
FROM department, employee
WHERE department.dnumber = employee.dno and
        dname = 'Research';
```

Special joins:

Inner joins and a Where clause

this example lists employees working on a project that is not where their department is located

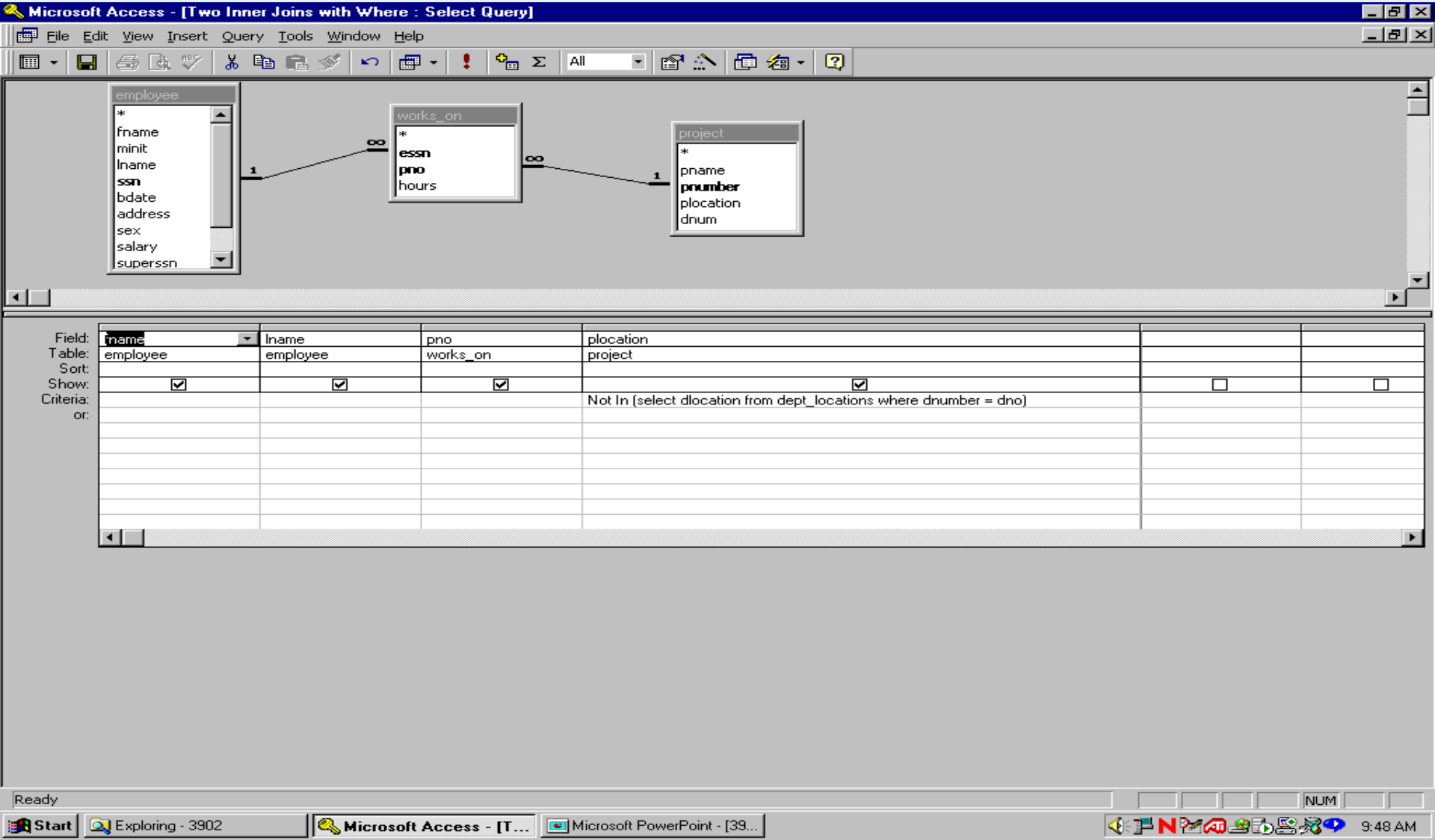
```
SELECT employee.fname, employee.lname, works_on.pno, project.plocation
FROM
    project INNER JOIN
    (employee INNER JOIN works_on ON employee.ssn = works_on.essn)
    ON project.pnumber = works_on.pno
WHERE
plocation not in (select dlocation from dept_locations where dnumber = dno);
```

Comparison:

```
SELECT employee.ssn, employee.fname, employee.lname,
        project.pnumber, project.plocation
FROM employee, project, works_on
WHERE
employee.ssn = works_on.essn and
project.pnumber = works_on.pno and
plocation NOT IN
        (SELECT dlocation FROM dept_locations WHERE
        dnumber=employee.dno);
```

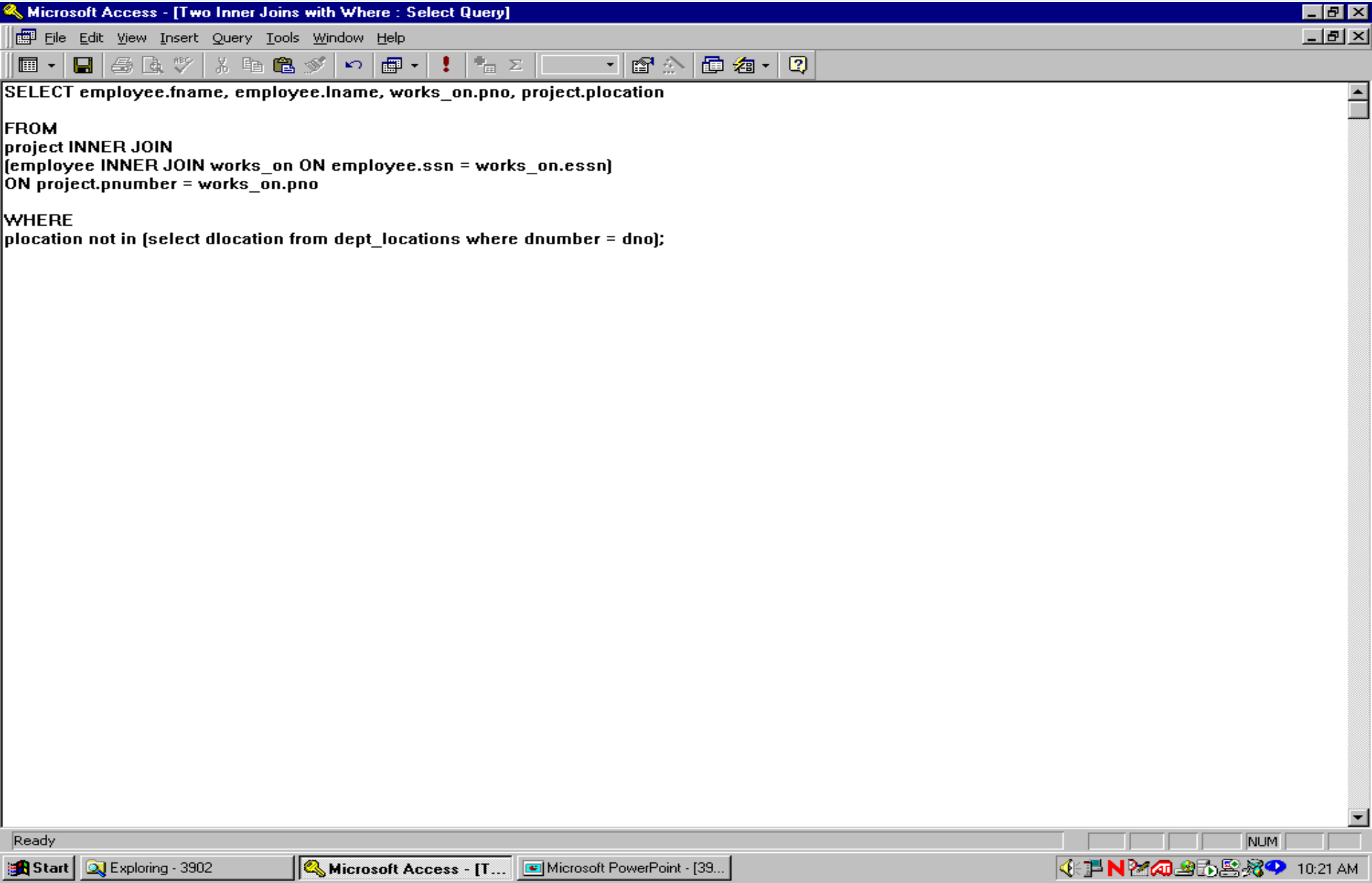
SQL

The design view was reduced to these tables and relationships. Access automatically incorporated the inner joins - see the SQL view



SQL

the SQL view



Update statements: pages 212-5

- Insert
- Update
- Delete

```
INSERT INTO employee ( fname, lname, ssn, dno )
VALUES ( "Joe", "Smith", 909, 1);
```

Note that Access changes the above to read:
INSERT INTO employee (fname, lname, ssn, dno)
SELECT "Joe", "Smith", 909, 1;

```
UPDATE employee SET salary = 100000
WHERE ssn=909;

DELETE FROM employee WHERE ssn=909;
```

Views: pages 215-9

- Use a Create View command
  - essentially a select specifying the data that makes up the view
  - Create View Enames as select lname, fname from employee

```
CREATE VIEW     Enames (lname, fname)
AS SELECT       LNAME, FNAME
    FROM        EMPLOYEE
```

```
CREATE VIEW     DEPT_INFO (DEPT_NAME,
                           NO_OF_EMPS,
                           TOTAL_SAL)
AS SELECT       DNAME, COUNT(*), SUM(SALARY)
FROM            DEPARTMENT, EMPLOYEE
WHERE           DNUMBER = DNO
GROUP BY        DNAME;
```

- Views are very common in business systems
  - users' view of data is simplified
    - Create View EmpProj as select lname, fname, pno from employee inner join …… *As complex as needed*
  - a form of security - user sees only the data he/she needs to
  - if the primary key is preserved, updates to a base table through a view is possible

Other SQL capabilities

- **Assertions** can be used for some constraints
  - e.g. Create Assertion ... ...

Executed and enforced by DBMS

Constraint: The salary of an employee must not be greater than the salary of the manager of the department that the employee works for.

```
CREATE ASSERTION salary_constraint
CHECK (NOT EXISTS (SELECT * FROM employee e,
                                  employee m, department d
where e.salary > m.salary and e.dno=d.dnumber and
        d.mgrssn=m.ssn));
```

- **Assertions in old version of SQL**
  - **Assert**
  - **trigger**

Assert statement

```
Assert SALARY_Constraint on employee e,
                              employee m, department d:
not (e.salary > m.salary and e.dno=d.dnumber and
d.mgrssn=m.ssn);
```

Trigger

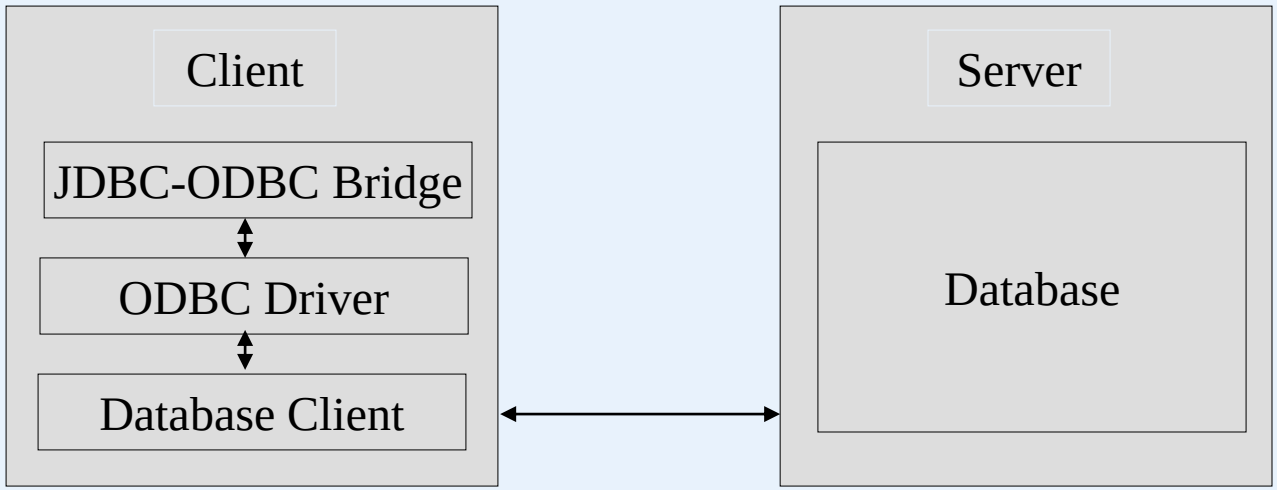
```
DEFINE trigger SALARY_TRIGGER on
          employee e, employee m, department d:
 (e.salary > m.salary and e.dno=d.dnumber and
d.mgrssn=m.ssn
  ACTION_PROCEDURE inform_manager(d.mgrssn);
```

- **Security**: Grant and Revoke are used to specify user privileges
  - Grant select, update on Employee to Diane;
  - Revoke update, delete on Works_on from Jim;
- **Embedded SQL**: SQL can be placed within 3GL programs
- **Transactions**: SQL systems implement the ACID properties

To develop a database application, **JDBC** or **ODBC** should be used.

JDBC – JAVA Database Connectivity

ODBC – Open Database Connectivity



Connection to a database:

1. Loading driver class

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

2. Connection to a database

```
String url = "jdbc:odbc:<databaseName>";

Connction con =
DriverManager.getConnection(url, <userName>,
<password>)
```

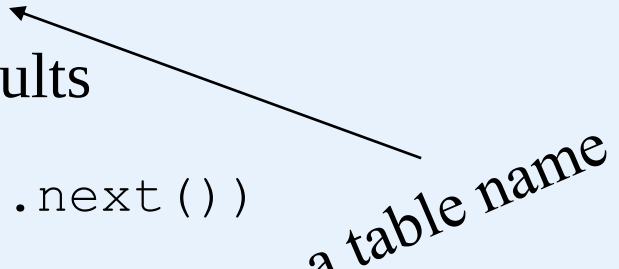
3. Sending SQL statements

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

ResultSet rs = stmt.executeQuery("SELECT *
FROM Information WHERE Balance >= 5000");
```

4. Getting results

```
while (rs.next())
    { …
    }
```



```
import java.sql.*;

public class DataSourceDemo1
{ public static void main(String[] args)
  { Connection con = null;
    try
    {//load driver class
     Class.forName{“sun.jdbs.odbs.JdbcOdbcDriver”);

     //data source
     String url = “jdbs:odbc:Customers”;

     //get connection
     con = DriverManager.getConnection(url,
     “sa”, “ “)
```

```
//create SQL statement
Statement stmt = con.createStatement();

//execute query
Result rs = stmt.executeQuery("SELECT *
FROM Information WHERE Balance >= 5000");

String firstName, lastName;
Date birthDate;
float balance;
int accountLevel;
```

```
      while(rs.next())
      {firstName = rs.getString("FirstName");
       lastName = rs.getString("lastName");
       balance = rs.getFloat("Balance");

       System.out.println(firstName + " " +
       lastName + ", balance = " + balance);
      }
     }
     catch(Exception e)
     {e.printStackTrace();}
     finally
     {try{con.close();}
      catch(Exception e){ }
     }
   }
}
```

Programming in a dynamical environment:

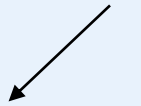
Disadvantage of *DataSourceDemo1*:

If the JDBC-ODBC driver, database, user names, or password are changed, the program has to be modified.

Solution:

Configuration file:

file name: datasource.config



config.driver=sun.jdbc.odbc.JdbcOdbcDriver
config.protocol=jdbc
config.subprotocol=odbc
config.dsname=Customers
config.username=sa
config.password=… …

```
<property> = <property value>
```

```
config - datasource name
```

```
import java.sql.*;
import java.io.*;
import java.util.Properties;

public class DatabaseAccess
{ private String configDir;
  //directory for configuration file
  private String dsDriver = null;
  private String dsProtocol = null;
  private String dsSubprotocol = null;
  private String dsName = null;
  private String dsUsername = null;
  private String dsPassword = null;
```

```
public DatabaseAccess(String configDir)
{ this.configDir = configDir; }

public DatabaseAccess()
{ this(“.”); }

//source: data source name
//configFile: source configuration file

public Connection getConnection(String source,
String configFile) throws SQLException, Exception
{ Connection con = null;

  try
  {Properties prop = loadConfig(ConfigDir, ConfigFile);
```

```
getConnection(“config”,
              “datasource.config”);
```

```
  if (prop != null)
  {dsDriver = prop.getProperty(source + ".driver");
   dsProtocol = prop.getPropert(source + ".protocol");
   dsSubprotocol = prop.getPropert(source +
   ".subprotocol");
   if (dsName == null)
     dsName = prop.getProperty(source +
     ".dsName");
   if (dsUsername == null)
dsUsername = prop.getProperty(source +
".username");
   if (dsPassword == null)
     dsPassword = prop.getProperty(source +
".password");
```

```
    //load driver class
    Class.forName(dsDriver);

    //connect to data source
    String url = dsProtocol + ":" + dsSubprotocol + ":"
    + dsName;
    con = DriverManager.getConnection(url, dsUsername,
    dsPassword)
  }
  else
    throw new Exception("* Cannot find property file +
    configFile);

  return con;
}
catch (ClassNotFoundException e)
{ throw new Exception("* Cannot find driver class " +
  dsDriver + "!"); }
}
```

```
//dir: directory of configuration file
//filename: file name
public Properties loadConfig(String dir, String filename)
throws Exception
{ File inFile = null;
  Properties prop = null;

  try
  { inFile = new File(dir, filename);
    if (inFile.exists()
    {  prop = new Properties();
       prop.load(new FileInputStream(inFile));
    }
    else throw new Exception("* Error in finding " +
              inFile.toString());
  }
  finally {return prop;}
  }
}
```

Using class DatabaseAccess, DataSourceDemo1 should be modified a little bit:

```
DatabaseAccess db = new databaseAccess();

con = db.getConnection("config",
"datasource.config");
```

Database updating:

```
import java.sql.*;

public class UpdateDemo1
{ public static void main(String[] args)
  { Connection con = null;
    try
    {
     //get connection
     Databaseaccess db = new DatabaseAccess();
     con = db.getConnection("config",
     "datasource.config");
```

```
      //execute update
      Statement stmt = con.CreateStatement();
      int account = stmt.executeUpdate("UPDATE
      Information SET Accountlevl = 2 WHERE
      Balance >= 50000");
      System.out.println(account + " record has been
  updated");

      //execute insert
      account = stmt.executeUpdate("INSERT INTO
      Information VALUE ('David', 'Feng', '05/05/1975',
      2000, 1)");
      System.out.println(account + " record has been
      inserted");
    }
    catch (Exception e) {e.printStackTrace(); }
    finally {try{con.close(); catch(Exception e){ }}
  }
}
```

Database updating:

```
import java.sql.*;

public class UpdateDemo2
{ public static void main(String[] args)
  { Connection con = null;
    try
    {
     //get connection
     Databaseaccess db = new DatabaseAccess();
     con = db.getConnection("config",
     "datasource.config");

     //create SQL statement and make the object of
     //ResultSet scrollable and updatable
```

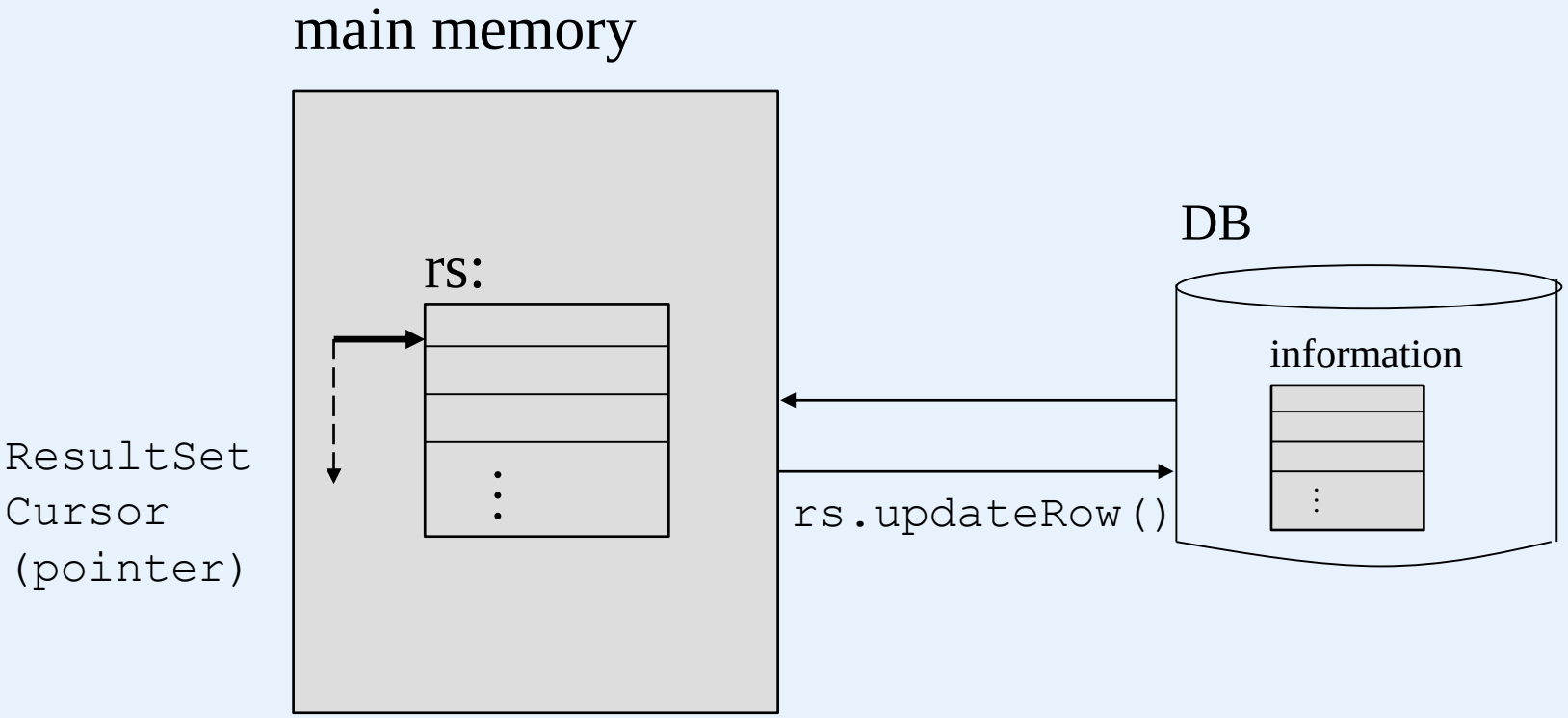
```
//create SQL statement and make the object of
//ResultSet scrollable and updatable

Statement stmt = con.CreateStatement(
ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_UPDATABLE);

//execute query
ResultSet rs = stmt.executeQuery("SELECT * FROM
Information WHERE Balance >= 50000");

while (rs.next())
{ rs.updateInt("AccountLevel", 2);
  rs.updateRow();
}
```

```
      //insert a new row
      rs.moveToInsertRow();
      rs.updateString("FirstName", "David");
      rs.updateString("LastName", "Feng");
      rs.updateDate("Birthdate", Date.ValueOf("1975-05-05")
      rs.updateFloat("Balance", 20000);
      rs.updateInt("AccountLevel", 1);
      rs.insertRow();
    }
    catch (Exception e) {e.printStackTrace(); }
    finally {try{con.close(); catch(Exception e){ }}
  }
}
```




```
ResultSet.TYPE_FORWARD_ONLY
ResultSet.CONCUR_UPDATABLE
```

Get Metadata

To get metadata of a databases, using the following methods:

```
ResultSet getTables(String catalog, String
schemaPattern, String tableNamePattern,
String[] types)
```

types is an array possibly containing “TABLE”, “SYSTEM TABLE”, or “VIEW’.

```
ResultSet getColumns(String catalog, String
schemaPattern, String tableNamePattern,
String columnNamePattern)
```

```
import java.sql.*;

public class MetaDataDemo
{ public static void main(String[] args)
  { Connection con = null;
    try
    {
     //get connection
     Databaseaccess db = new DatabaseAccess();
     con = db.getConnection("config",
     "datasource.config");

     DatabaseMetaData metaData = con.getMetadata();
     ResultSet tables = metaData.getTables(null,
     null, null, New String[] {"TABLE"});
```

```
      ResultSet columnNames;
      String tableName, columnName;

      while (tables.next())
      { tableName = tables.getString("TABLE_NMAE");
        columnNames = metaData.getColumn(null, null,
        tableName, null);

        System.out.println("Table: " + tableName);
        while columnNames.next())
        { columnName = columnName.getString("COLUMN_NAME");
          System.out.println(columnName);
        }
    }
    catch (Exception e) {e.printStackTrace(); }
    finally {try{con.close(); catch(Exception e){ }}
  }
}
```

Prepared Statement

```
String sql = "SELECT * FROM Information
WHERE Age > ? AND Balance > ?";

PreparedStatement pstmt = con.preparedStatement(sql)

pstmt.setInt(1, 30);

pstmt.setInt(2, 50000);
```

```
import java.sql.*;

public class PreparedStatementDemo
{ public static void main(String[] args)
  { Connection con = null;
    try
    {
     Databaseaccess db = new DatabaseAccess();
     con = db.getConnection("config",
     "datasource.config");
     String sql = "SELECT * FROM Information WHERE
     Balance > ? AND BirthDate <= ?";
     PreparedStatement pstmt =
      con.preparedStatement(sql)
     pstmt.setInt(1, 50000);
     pstmt.setInt(2, Date.valueOf("1970-01-01"));
```

```
      ResultSet rs = pstmt.executeQuery();

      String firstName, lastName;
      Date birthDate;
      float balance;
      int accountLevel;

      while (rs.next())
      { firstName = rs.getString("FirstName");
        lastName = rs.getString("LastName");
        balance = rs.getfloat("Balance");
        System.out.println(firstName + " " + lastName
        + ", balance = " + balance);
      }
      catch (Exception e) {e.printStackTrace(); }
      finally {try{con.close(); catch(Exception e){ }}
  }
}
```

```
Int j[] = {50,000, 60,700, 100,000, 45,000}
String sql = "SELECT * FROM Information
WHERE Balance = ?";

for (i = 1; <= 4) {
      PreparedStatement pstmt =
      con.preparedStatement(sql)
      pstmt.setInt(1, j[i]);
      ResultSet rs = pstmt.executeQuery();
}
```

Transaction

A transaction is a sequence of instructions, which is considered to be a unit. If part of an transaction fails, the whole transaction will be rolled back as if it is not executed at all.

The result of a transaction is stored only after the command “commit” is executed.

void setAutoCommit(boolean autoCommit);

void commit();

void rollback();

only for advanced database course

Transaction isolation level:

TRANSACTION_NONE

TRANSACTION_READ_UNCOMMITED (dirty read)

TRANSACTION_READ_COMMITED (dirty read is not allowed)

TRANSACTION_REPEATED_READ

TRANSACTION_SERIALIZABLE

```
int getTransactionIsolation();

void setTransactionIsolation(int level);
```

```
import java.sql.*;

public class BatchUpdateDemo
{ public static void main(String[] args)
  { Connection con = null;
    try
    {
     Databaseaccess db = new DatabaseAccess();
     con = db.getConnection("config",
     "datasource.config");
     DatabaseMetaData dbmd = con.getMetaData();
     if (dbmd.supportsBatchUpdate())
     { //diable auto commit
      con.setAutoCommit(false);
```

```
//add batch update
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO Information VALUES
('A', 'Chen', '05/05/1975', 20000, 1)");
stmt.addBatch("INSERT INTO Information VALUES
('A', 'Chen', '06/05/1975', 30000, 1)");
stmt.addBatch("INSERT INTO Information VALUES
('A', 'Chen', '07/05/1975', 40000, 1)");

//execute batch update
int[] updateCounts = stmt.executeBatch();
con.commit();
con.setAutoCommit(true);
}
else {System.out.println("Driver does not support
        batch updates.");
}
}
```

```
      catch (BatchUpdateException be)
      {int[] updateCounts = be.getUpdateCounts();
       for (int i = 0; i < updateCounts.length; i++)
       {System.out.println("Batch Update " + i + ": "
             + updateCounts[i]);
       }
      catch (Exception e) {e.printStackTrace(); }
      finally {try{con.close(); catch(Exception e){ }}
  }
}
```