Advanced Databases

Winter Term 2010/11

Prof. Dr. Dietmar Seipel
University of Würzburg
## Contents

1 ER Modelling
   1.1 The ER Model of the COMPANY Database ........................................... 2
   1.2 Mapping the ER Model to a Relational Model ........................................ 5

---

Prof. Dr. Dietmar Seipel
2 The Relational Query Language SQL

2.1 Data Definition in SQL ................................. 13
2.2 Queries in SQL ........................................... 21
2.3 Update Statements in SQL ............................... 57
2.4 Views in SQL ............................................. 63
2.5 Indexes in SQL ........................................... 67
2.6 Java Database Connectivity (JDBC) .................. 69
1 ER Modelling

• Initially, a relational database is designed using the Entity–Relationship Model (ER), which models similar objects of the real world using entity types and relationships between them as relationship types. Entities and relationships are described using attributes.

• A graphical representation is used for ER diagrams. Similar concepts are nowadays also used for object–oriented modelling with the Unified Modelling Language (UML).

• There exists a generic mapping from the ER model to the relational data model, which maps both entity as well as relationship types to relation schemas accompanied by primary and foreign key constraints.
1.1 The ER Model of the COMPANY Database

The COMPANY database contains the following entity types:

- EMPLOYEE and DEPENDENT,
- DEPARTMENT, and
- PROJECT.

The relationship types represent the following:

- the supervisor and the dependents of the employees are given by SUPERVISION and DEPENDENTS_OF;

  the weak entity type DEPENDENT is made unique by the relationship type DEPENDENTS_OF and the entity type EMPLOYEE;

- the employees and the managers of the departments are given by WORKS_FOR and MANAGES; the functionalities and the existency constraints require that every employee works for exactly one
department and that every department must have exactly one manager; an employee can manage at most one department;

- WORKS_ON gives the employees working for a project, and CONTROLS gives the responsible department; the functionalities and the existency constraints require that every employee must work for at least one project and that every project must have exactly one responsible department.
1.2 Mapping the ER Model to a Relational Model

In EMPLOYEE, the components of the complex attribute NAME from the ER diagram become 3 separate attributes.

The 1:n relationship types WORKS_F FOR and SUPERVISION are integrated as foreign keys DNO and SUPERSSN, respectively, in EMPLOYEE.

<table>
<thead>
<tr>
<th>FNAME</th>
<th>MINIT</th>
<th>LNAME</th>
<th>SSN</th>
<th>BDATE</th>
<th>ADDRESS</th>
<th>SEX</th>
<th>SALARY</th>
<th>SUPERSSN</th>
<th>DNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>B</td>
<td>Smith</td>
<td>444444444</td>
<td>1955-01-09</td>
<td>731 Fondren, Houston, TX</td>
<td>M</td>
<td>30000</td>
<td>222222222</td>
<td>5</td>
</tr>
<tr>
<td>Franklin</td>
<td>T</td>
<td>Wong</td>
<td>222222222</td>
<td>1945-12-08</td>
<td>638 Voss, Houston, TX</td>
<td>M</td>
<td>40000</td>
<td>111111111</td>
<td>5</td>
</tr>
<tr>
<td>Alicia</td>
<td>J</td>
<td>Zelaya</td>
<td>777777777</td>
<td>1958-07-19</td>
<td>3321 Castle, Spring, TX</td>
<td>F</td>
<td>25000</td>
<td>333333333</td>
<td>4</td>
</tr>
<tr>
<td>Jennifer</td>
<td>S</td>
<td>Wallace</td>
<td>333333333</td>
<td>1931-06-20</td>
<td>291 Berry, Bellaire, TX</td>
<td>F</td>
<td>43000</td>
<td>111111111</td>
<td>4</td>
</tr>
<tr>
<td>Ramesh</td>
<td>K</td>
<td>Narayan</td>
<td>555555555</td>
<td>1952-09-15</td>
<td>975 Fire Oak, Humble, TX</td>
<td>M</td>
<td>38000</td>
<td>222222222</td>
<td>5</td>
</tr>
<tr>
<td>Joyce</td>
<td>A</td>
<td>English</td>
<td>666666666</td>
<td>1962-07-31</td>
<td>5631 Rice, Houston, TX</td>
<td>F</td>
<td>25000</td>
<td>222222222</td>
<td>5</td>
</tr>
<tr>
<td>Ahmad</td>
<td>V</td>
<td>Jabbar</td>
<td>888888888</td>
<td>1959-03-29</td>
<td>980 Dallas, Houston, TX</td>
<td>M</td>
<td>25000</td>
<td>333333333</td>
<td>4</td>
</tr>
<tr>
<td>James</td>
<td>E</td>
<td>Borg</td>
<td>111111111</td>
<td>1927-11-10</td>
<td>450 Stone, Houston, TX</td>
<td>M</td>
<td>55000</td>
<td>NULL</td>
<td>1</td>
</tr>
</tbody>
</table>
The departments, their employees, and the supervisors of the employees:

**Headquarters**
- 1: James Borg

**Research**
- 222222222
  - 5: Franklin Wong
  - 222222222
    - 5: John Smith
    - 555555555
      - 5: Ramesh Narayan
      - 666666666
        - 5: Joyce English

**Administration**
- 333333333
  - 4: Jennifer Wallace
  - 777777777
    - 4: Alicia Zelaya
    - 888888888
      - 4: Ahmad Jabbar
The 1:1 relationship type MANAGES is integrated as a foreign key MGRSSN in DEPARTMENT together with the describing attribute MGRSTARTDATE.

The multi–valued attribute Locations of the entity type DEPARTMENT yields a separate table DEPT_LOCATIONS.

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>DEPT_LOCATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNAME</td>
<td>DNUMBER</td>
</tr>
<tr>
<td>Headquarters</td>
<td>1</td>
</tr>
<tr>
<td>Administration</td>
<td>4</td>
</tr>
<tr>
<td>Research</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Sugarland</td>
</tr>
<tr>
<td>5</td>
<td>Houston</td>
</tr>
</tbody>
</table>
### WORKS_ON

<table>
<thead>
<tr>
<th>ESSN</th>
<th>PNO</th>
<th>HOURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>111111111</td>
<td>20</td>
<td>NULL</td>
</tr>
<tr>
<td>222222222</td>
<td>2</td>
<td>10.0</td>
</tr>
<tr>
<td>222222222</td>
<td>3</td>
<td>10.0</td>
</tr>
<tr>
<td>333333333</td>
<td>30</td>
<td>20.0</td>
</tr>
<tr>
<td>333333333</td>
<td>20</td>
<td>15.0</td>
</tr>
<tr>
<td>444444444</td>
<td>1</td>
<td>32.5</td>
</tr>
<tr>
<td>444444444</td>
<td>2</td>
<td>7.5</td>
</tr>
<tr>
<td>555555555</td>
<td>3</td>
<td>40.0</td>
</tr>
<tr>
<td>666666666</td>
<td>1</td>
<td>20.0</td>
</tr>
<tr>
<td>666666666</td>
<td>2</td>
<td>20.0</td>
</tr>
<tr>
<td>777777777</td>
<td>30</td>
<td>30.0</td>
</tr>
<tr>
<td>777777777</td>
<td>10</td>
<td>10.0</td>
</tr>
<tr>
<td>888888888</td>
<td>10</td>
<td>35.5</td>
</tr>
<tr>
<td>888888888</td>
<td>30</td>
<td>5.0</td>
</tr>
</tbody>
</table>

*Diagram showing connections between ESSNs and locations.*
The 1:n relationship type **CONTROLS** between **DEPARTMENT** and **PROJECT** is integrated as the foreign key **DNUM** in **PROJECT**.

<table>
<thead>
<tr>
<th>PNAME</th>
<th>PNUMBER</th>
<th>PLOCATION</th>
<th>DNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductX</td>
<td>1</td>
<td>Bellaire</td>
<td>5</td>
</tr>
<tr>
<td>ProductY</td>
<td>2</td>
<td>Sugarland</td>
<td>5</td>
</tr>
<tr>
<td>ProductZ</td>
<td>3</td>
<td>Houston</td>
<td>5</td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
<td>Stafford</td>
<td>4</td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
<td>Houston</td>
<td>1</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>Stafford</td>
<td>4</td>
</tr>
</tbody>
</table>

Every project is located at one of the locations of its controlling departments.
The weak entity type `DEPENDENT` becomes unique by means of the foreign key ESSN, which references the key SSN of the entity type `EMPLOYEE`.

<table>
<thead>
<tr>
<th>ESSN</th>
<th>DEPENDENT_NAME</th>
<th>SEX</th>
<th>BDATE</th>
<th>RELATIONSHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2222222222</td>
<td>Alice</td>
<td>F</td>
<td>1976-04-05</td>
<td>DAUGHTER</td>
</tr>
<tr>
<td>2222222222</td>
<td>Theodore</td>
<td>M</td>
<td>1973-10-25</td>
<td>SON</td>
</tr>
<tr>
<td>2222222222</td>
<td>Joy</td>
<td>F</td>
<td>1948-05-03</td>
<td>SPOUSE</td>
</tr>
<tr>
<td>3333333333</td>
<td>Abner</td>
<td>M</td>
<td>1932-02-29</td>
<td>SPOUSE</td>
</tr>
<tr>
<td>4444444444</td>
<td>Michael</td>
<td>M</td>
<td>1978-01-01</td>
<td>SON</td>
</tr>
<tr>
<td>4444444444</td>
<td>Alice</td>
<td>F</td>
<td>1978-12-31</td>
<td>DAUGHTER</td>
</tr>
<tr>
<td>4444444444</td>
<td>Elizabeth</td>
<td>F</td>
<td>1957-05-05</td>
<td>SPOUSE</td>
</tr>
</tbody>
</table>
Referential integrity constraints displayed on the COMPANY relational database schema
2 The Relational Query Language SQL

The relational data model was first proposed by E.F. Codd in 1970.

- The Structured Query Language (SQL) is used for accessing the data. Internally, SQL queries are evaluated using the operations of the relational algebra.

- There exist statements for creating and destroying (CREATE, DROP) databases and tables, respectively, for inserting into and for updating tables (INSERT, DELETE, UPDATE), and for the data access (SELECT). A SELECT statement can join several tables, and it can form groups and aggregate within them.

- SQL can be called from programming languages like Java (Java Database Connectivity, JDBC).
2.1 Data Definition in SQL

The ER design can be translated into SQL CREATE TABLE–Statements:

1. For every attribute of a table also a data type is given, and possibly a NOT NULL constraint and a DEFAULT value.

2. A combination $K$ of attributes is specified as the primary key by **PRIMARY KEY** $(K)$ with an automatic NOT NULL constraint. The primary key is used for organizing the data records on the secondary storage, and automatically an index is created.

3. Further combinations $X$ of attributes can be made to secondary keys by **UNIQUE** $(X)$. Here, NOT NULL constraints have to be specified explicitly, if desired.

4. Foreign key constraints from one combination $X$ of attributes to the primary key $K$ of a table $T$ are specified by **FOREIGN KEY** $(X)$ **REFERENCES** $T(K)$. 
Creating a database and connecting with it:

```
CREATE DATABASE COMPANY;
USE COMPANY;
```

Creating a 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,
);```
The NOT NULL constraint for DNO implements the existency constraint from employess to departments from WORKS_FOR. The inverse existency constraint cannot be represented in a CREATE TABLE statement.

By UNIQUE (FNAME, MINIT, LNAME, BDATE), one could make the combination of the name and the birthdate to a further, secondary key.
Creating the complete COMPANY database

CREATE TABLE DEPARTMENT (  
    DNAME VARCHAR(15) NOT NULL,  
    DNUMBER INT NOT NULL,  
    MGRSSN CHAR(9) NOT NULL,  
    MGRSTARTDATE DATE,  
    PRIMARY KEY (DNUMBER),  
    UNIQUE (DNAME)  
    FOREIGN KEY (MGRSSN)  
    REFERENCES EMPLOYEE (SSN) );

The NOT NULL constraint for MGRSSN implements the existency constraint from MANAGES, which requires that a department must have a manager.
CREATE TABLE DEPT_LOCATIONS (  
  DNUMBER    INT        NOT NULL,  
  DLOCATION  VARCHAR(15) NOT NULL,  
  PRIMARY KEY (DNUMBER, DLOCATION),  
  FOREIGN KEY (DNUMBER)  
    REFERENCES DEPARTMENT(DNUMBER)  
  ON DELETE CASCADE  ON UPDATE CASCADE );

The foreign key constraint a combined with a trigger,  
that is activated by deletions and updates in DEPARTMENT.
CREATE TABLE PROJECT (  
PNAME VARCHAR(15) NOT NULL,  
PNUMBER INT NOT NULL,  
PLOCATION VARCHAR(15),  
DNUM INT NOT NULL,  
PRIMARY KEY (PNUMBER),  
UNIQUE (PNAME),  
FOREIGN KEY (DNUM)  
    REFERENCES DEPARTMENT (DNUMBER) );

The NOT NULL constraint for DNUM implements the existency constraint from CONTROLS, which requires that a project must have a controlling department.
CREATE TABLE WORKS_ON (  
    ESSN   CHAR(9) NOT NULL,  
    PNO    INT NOT NULL,  
    HOURS  DECIMAL(3,1) NOT NULL,  
    PRIMARY KEY (ESSN, PNO),  
    FOREIGN KEY (ESSN)  
    REFERENCES EMPLOYEE(SSN),  
    FOREIGN KEY (PNO)  
    REFERENCES PROJECT(PNUMBER) );

The existency constraints from WORKS_FOR cannot be implemented by CREATE TABLE statements.
```sql
CREATE TABLE DEPENDENT (
    ESSN CHAR(9) NOT NULL,
    DEPENDENT_NAME VARCHAR(15) NOT NULL,
    SEX CHAR,
    BDATE DATE,
    RELATIONSHIP VARCHAR(8),
    PRIMARY KEY (ESSN, DEPENDENT_NAME),
    FOREIGN KEY (ESSN)
        REFERENCES EMPLOYEEE(SSID) );
```
2.2 Queries in SQL

**SELECT–FROM–WHERE Block:**

Compared to relational algebra, the SELECT part represents a projection on the interesting attributes, the FROM part represents the join of the listed tables, and the WHERE part contains the selection and join conditions.

\[
\text{SELECT} \quad \text{<attribute list>} \quad \rightarrow \quad \text{resultat attributes} \\
\text{FROM} \quad \text{<table list>} \quad \rightarrow \quad \text{queried tables} \\
\text{WHERE} \quad \text{<condition>} \quad \rightarrow \quad \text{conditions for the result tuples}
\]

Often, joins are computed over *key/foreign key* conditions; this helps finding suitable join conditions. Furthermore, the join partners for tuples from the table with the foreign key can be found quickly in the table with the key.
Query 0 (Selection and Projection)

Find the birthdate and the address of the employee with the name “John B. Smith”.

\[
\begin{align*}
\text{SELECT} & \quad \text{BDATE, ADDRESS} \\
\text{FROM} & \quad \text{EMPLOYEE} \\
\text{WHERE} & \quad \text{FNAME} = 'John' \ \text{AND} \ \text{MINIT} = 'B' \ \text{AND} \ \text{LNAME} = 'Smith'
\end{align*}
\]

Result:

<table>
<thead>
<tr>
<th>BDATE</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1955-01-09</td>
<td>731 Fondren, Houston, TX</td>
</tr>
</tbody>
</table>

In the relational algebra:

\[
\Pi_{\text{BDATE, ADDRESS}}(\sigma_{\text{FNAME} = 'John' \ \text{AND} \ \text{MINIT} = 'B' \ \text{AND} \ \text{LNAME} = 'Smith'}(\text{EMPLOYEE})).
\]
Query (Projektion and Join)

For all employees – given by first and last name –, determine their projects – given by the project name – and the number of hours worked on the project.

\[
\text{SELECT FNAME, LNAME, PNAME, HOURS FROM EMPLOYEE, WORKS_ON, PROJECT WHERE SSN=ESSN AND PNO=PNUMBER}
\]

Every table name in the FROM part represents a tuple.

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FNAME</td>
<td>. .</td>
</tr>
<tr>
<td>LNAME</td>
<td>SSN</td>
</tr>
<tr>
<td>. .</td>
<td>. .</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORKS_ON</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSN</td>
<td>PNO</td>
</tr>
<tr>
<td>HOURS</td>
<td>. .</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJECT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PNAME</td>
<td>PNUMBER</td>
</tr>
<tr>
<td>. .</td>
<td>. .</td>
</tr>
</tbody>
</table>

In the relational algebra:

\[
\Pi_{\text{FNAME, LNAME, PNAME, HOURS}}(\text{EMPLOYEE} \bowtie_{\text{SSN}=\text{ESSN}} \text{WORKS_ON} \bowtie_{\text{PNO}=\text{PNUMBER}} \text{PROJECT}).
\]
For every triple of tuples satisfying the **WHERE** condition, a resulting tuple can be constructed by projection on the attributes in the **SELECT** part.

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>PNAME</th>
<th>HOURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>ProductX</td>
<td>32.5</td>
</tr>
<tr>
<td>John</td>
<td>Smith</td>
<td>ProductY</td>
<td>7.5</td>
</tr>
<tr>
<td>Ramesh</td>
<td>Narayan</td>
<td>ProductZ</td>
<td>40.0</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
<td>ProductX</td>
<td>20.0</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
<td>ProductY</td>
<td>20.0</td>
</tr>
<tr>
<td>Franklin</td>
<td>Wong</td>
<td>ProductY</td>
<td>10.0</td>
</tr>
<tr>
<td>Franklin</td>
<td>Wong</td>
<td>ProductZ</td>
<td>10.0</td>
</tr>
<tr>
<td>Alicia</td>
<td>Zelaya</td>
<td>Newbenefits</td>
<td>30.0</td>
</tr>
<tr>
<td>Alicia</td>
<td>Zelaya</td>
<td>Computerization</td>
<td>10.0</td>
</tr>
<tr>
<td>Ahmad</td>
<td>Jabbar</td>
<td>Computerization</td>
<td>35.5</td>
</tr>
<tr>
<td>Ahmad</td>
<td>Jabbar</td>
<td>Newbenefits</td>
<td>5.0</td>
</tr>
<tr>
<td>Jennifer</td>
<td>Wallace</td>
<td>Newbenefits</td>
<td>20.0</td>
</tr>
<tr>
<td>Jennifer</td>
<td>Wallace</td>
<td>Reorganization</td>
<td>15.0</td>
</tr>
<tr>
<td>James</td>
<td>Borg</td>
<td>Reorganization</td>
<td>NULL</td>
</tr>
</tbody>
</table>
With binary joins in the relational algebra:
\[
\Pi_{\text{FNAME}, \text{LNAME}, \text{PNAME}, \text{HOURS}}(\text{EMPLOYEE} \bowtie_{\text{SSN} = \text{ESSN}} (\text{WORKS\_ON} \bowtie_{\text{PNO} = \text{PNUMBER}} \text{PROJECT})).
\]

Corresponding operator tree:

\[
\mathcal{O}_1:
\]

\[
\Pi_{\text{FNAME}, \text{LNAME}, \text{PNAME}, \text{HOURS}} \quad \text{EMPLOYEE} \bowtie_{\text{SSN} = \text{ESSN}} \quad \text{WORKS\_ON} \bowtie_{\text{PNO} = \text{PNUMBER}} \quad \text{PROJECT}
\]
**Query 1 (Selection, Projection and Join)**

Find the name and address of all employees working for the “Research” department.

```sql
SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNUMBER=DNO
```

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>731 Fondren, Houston, TX</td>
</tr>
<tr>
<td>Franklin</td>
<td>Wong</td>
<td>638 Voss, Houston, TX</td>
</tr>
<tr>
<td>Ramesh</td>
<td>Narayan</td>
<td>975 Rice, Houston, TX</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
<td>5631 Rice, Houston, TX</td>
</tr>
</tbody>
</table>

In the relational algebra:

\[ \Pi_{\text{FNAME}, \text{LNAME}, \text{ADDRESS}}(\sigma_{\text{DNAME}='Research'}(\text{EMPLOYEE} \bowtie_{\text{DNUMBER}=\text{DNO}} \text{DEPARTMENT})) \].
Query Optimization: Push Selection into Join

\[ \Pi_{\text{FNAME}, \text{LNAME}, \text{ADDRESS}}(\text{EMPLOYEE} \bowtie_{\text{DNUMBER} = \text{DNO}} \sigma_{\text{DNAME} = 'Research'}(\text{DEPARTMENT})) \].

Corresponding operator tree:

\[ O_2: \]

\[ \Pi_{\text{FNAME}, \text{LNAME}, \text{ADDRESS}} \]

\[ \bowtie_{\text{DNUMBER} = \text{DNO}} \]

\[ \sigma_{\text{DNAME} = 'Research'} \]

\[ \text{EMPLOYEE} \]

\[ \text{DEPARTMENT} \]
Query 2

List the project number, the department number, and the last name, address and birthdate of the manager of the department for all projects located in Stafford.

```
SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN
AND PLOCATION='Stafford'
```

Result:

<table>
<thead>
<tr>
<th>PNUMBER</th>
<th>DNUM</th>
<th>LNAME</th>
<th>ADDRESS</th>
<th>BDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>Wallace</td>
<td>291 Berry, Bellaire, TX</td>
<td>1931-06-20</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>Wallace</td>
<td>291 Berry, Bellaire, TX</td>
<td>1931-06-20</td>
</tr>
</tbody>
</table>
**Ambiguous Attribute Names and Aliasing**

To disambiguate,

- an attribute name $A$ can be made clear by prefixing it with the relation name $R (R.A)$, and
- one can assign an alias $R'$ to a relation name $R$ ($R R'$, e.g. EMPLOYEE E).

This is for example necessary, if we join a relation with itself.

**Query 8**

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE E, EMPLOYEE S
WHERE E.SUPERSSN=S.SSN
```
The supervisor hierarchy:
### EMPLOYEE

| FNAME  | MINIT | LNAME  | SSN      | ... | SUPERSSN  | ...
|--------|-------|--------|----------|-----|-----------|-----
| John   | B     | Smith  | 444444444| ... | 222222222| ... |
| Franklin| T     | Wong   | 222222222| ... | 111111111| ... |

### Result:

<table>
<thead>
<tr>
<th>E.FNAME</th>
<th>E.LNAME</th>
<th>S.FNAME</th>
<th>S.LNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>Franklin</td>
<td>Wong</td>
</tr>
<tr>
<td>Franklin</td>
<td>Wong</td>
<td>James</td>
<td>Borg</td>
</tr>
<tr>
<td>Alicia</td>
<td>Zelaya</td>
<td>Jennifer</td>
<td>Wallace</td>
</tr>
<tr>
<td>Jennifer</td>
<td>Wallace</td>
<td>James</td>
<td>Borg</td>
</tr>
<tr>
<td>Ramesh</td>
<td>Narayan</td>
<td>Franklin</td>
<td>Wong</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
<td>Franklin</td>
<td>Wong</td>
</tr>
<tr>
<td>Ahmad</td>
<td>Jabbar</td>
<td>Jennifer</td>
<td>Wallace</td>
</tr>
</tbody>
</table>
Queries without WHERE Clause, Use of ’*’ (Wildcard)

Query 9 (Projektion)

Find all EMPLOYEE SSNs in the database.

```
SELECT SSN
FROM EMPLOYEE
```

Result:

<table>
<thead>
<tr>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>444444444</td>
</tr>
<tr>
<td>222222222</td>
</tr>
<tr>
<td>777777777</td>
</tr>
<tr>
<td>333333333</td>
</tr>
<tr>
<td>555555555</td>
</tr>
<tr>
<td>666666666</td>
</tr>
<tr>
<td>888888888</td>
</tr>
<tr>
<td>111111111</td>
</tr>
</tbody>
</table>

in the relational algebra: $\Pi_{\text{SSN}}(\text{EMPLOYEE})$
Query 1C (Selection)

\[
\text{SELECT } * \\
\text{FROM } \text{EMPLOYEE} \\
\text{WHERE } \text{DNO} = 5
\]

Result:

<table>
<thead>
<tr>
<th>FNAME</th>
<th>MINIT</th>
<th>LNAME</th>
<th>SSN</th>
<th>BDATE</th>
<th>ADDRESS</th>
<th>SEX</th>
<th>SALARY</th>
<th>SUPERSSN</th>
<th>DNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>B</td>
<td>Smith</td>
<td>444444444</td>
<td>1955-01-09</td>
<td>...</td>
<td>M</td>
<td>30000</td>
<td>222222222</td>
<td>5</td>
</tr>
<tr>
<td>Franklin</td>
<td>T</td>
<td>Wong</td>
<td>222222222</td>
<td>1945-12-08</td>
<td>...</td>
<td>M</td>
<td>40000</td>
<td>111111111</td>
<td>5</td>
</tr>
<tr>
<td>Ramesh</td>
<td>K</td>
<td>Narayan</td>
<td>555555555</td>
<td>1952-09-15</td>
<td>...</td>
<td>M</td>
<td>38000</td>
<td>222222222</td>
<td>5</td>
</tr>
<tr>
<td>Joyce</td>
<td>A</td>
<td>English</td>
<td>666666666</td>
<td>1962-07-31</td>
<td>...</td>
<td>F</td>
<td>25000</td>
<td>222222222</td>
<td>5</td>
</tr>
</tbody>
</table>

in the relational algebra: \( \sigma_{\text{DNO}=5}(\text{EMPLOYEE}) \)
**Tables as Sets**

**Query 11**

Retrieve the salary of all employees in the database.

```sql
SELECT SALARY
FROM EMPLOYEE
```

```sql
SELECT DISTINCT SALARY
FROM EMPLOYEE
```

If there are several employees with the same salary, then the first statement retrieves this salary multiply, while the second statement retrieves each salary only once (useful for computing the salary groups).
Query 4

Compute the list of all projects for which an employee with the last name “Smith” works or which are controlled by a department whose manager has the last name “Smith”.

( SELECT PNO
  FROM WORKS_ON, EMPLOYEE
  WHERE ESSN=SSN AND LNAME=’Smith’ )

UNION

( SELECT PNUMBER
  FROM PROJECT, DEPARTMENT, EMPLOYEE
  WHERE DNUM=DNUMBER
  AND MGRSSN=SSN AND LNAME=’Smith’ )
Nested Queries and Set Operations

Find the ESSN of all employees who work as many hours on a project as the employees with ESSN='444444444':

```
SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE (PNO, HOURS) IN (  
  SELECT PNO, HOURS  
  FROM WORKS_ON  
  WHERE ESSN='444444444' );
```

Here, the inner/outer variable ESSN refers to the inner/outer WORKS_ON.
Find all employees earning more than all employees of department 5:

```
SELECT  LNAME, FNAME
FROM     EMPLOYEE
WHERE    SALARY > ALL IN ( 
    SELECT  SALARY
    FROM     EMPLOYEE
    WHERE    DNO = 5 );
```

Also possible with \(<, =, <=, >=, <>,\) SOME, ANY
Query 12

Find the names of all employees having a dependent with the same first name and sex.

```sql
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE E
WHERE SSN IN (
    SELECT ESSN
    FROM DEPENDENT
    WHERE SSN = ESSN
    AND FNAME = DEPENDENT_NAME
    AND E.SEX = SEX )
```

The attribute SEX without prefix E in the inner SELECT statement refers to DEPENDENT.
An alternative formulation with a flat SELECT statement is:

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE E, DEPENDENT D
WHERE E.SSN = D.ESSN
AND E.FNAME = D.DEPENDENT_NAME
AND E.SEX = D.SEX
```
Explicit Sets and NULL Values

Query 13
Find the SSN of all employees working on one of the projects 1, 2, or 3.

SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE PNO IN (1, 2, 3)

Query 14
Find the names of all employees having no supervisor.

SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE SUPERSSN IS NULL
Query 3

Find the names of all employees working on all projectes controlled by department 5.

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE ( ( SELECT PNO
             FROM WORKS_ON
             WHERE SSN = ESSN )
       CONTAINS
       ( SELECT PNUMBER
         FROM PROJECT
         WHERE DNUM = 5 ) )
```

This query with **CONTAINS** can be evaluated using division in relational algebra.
**EXISTS**

**Query 6**

Find the names of all employees without dependents.

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE NOT EXISTS (
    SELECT *
    FROM DEPENDENT
    WHERE SSN = ESSN )
```
Renaming of Attributes and Explicit Joins

Query 8A

```
SELECT E.LNAME AS EMPLOYEE,
       S.LNAME AS SUPERVISOR
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.SUPERSSN = S.SSN
```

Result:

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>SUPERVISOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Wong</td>
</tr>
<tr>
<td>Wong</td>
<td>Borg</td>
</tr>
<tr>
<td>Zelaya</td>
<td>Wallace</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Query 1A

```
SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
WHERE DNAME='Research'
```

The join condition can be moved to the `FROM` part.

Query 1B

```
SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE NATURAL JOIN
     (DEPARTMENT AS D(DNAME, DNO, X, Y)))
WHERE DNAME='Research'
```

In the NATURAL JOIN, the join condition DNO=DNUMBER is implemented by renaming DNUMBER of DEPARTMENT using `AS` to DNO of EMPLOYEE.
Aggregate Functions and Grouping

Query 15

Compute the sum, maximum, minimum, and average of the salaries of all employees.

```
SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE
```

Null values (in SALARY) are neglected during aggregation.

Query 18

Determine the number of employees of the department ’Research’.

```
SELECT COUNT(*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME=’Research’
```
Query 19

Determine the number of different salaries in the database.

```
SELECT COUNT(DISTINCT SALARY)
FROM EMPLOYEE
```

Query 20

For all departments, determine the department number, the number of employees and the average salary of these employees.

```
SELECT DNO, COUNT(*) AS EMPS, AVG(SALARY)
FROM EMPLOYEE
GROUP BY DNO
```
Result of Query 20:

First, the employees are grouped according to their department.

<table>
<thead>
<tr>
<th>FNAME</th>
<th>MINIT</th>
<th>LNAME</th>
<th>SSN</th>
<th>...</th>
<th>SALARY</th>
<th>SUPERSSN</th>
<th>DNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>B</td>
<td>Smith</td>
<td>444444444</td>
<td></td>
<td>30000</td>
<td>222222222</td>
<td>5</td>
</tr>
<tr>
<td>Franklin</td>
<td>T</td>
<td>Wong</td>
<td>222222222</td>
<td></td>
<td>40000</td>
<td>111111111</td>
<td>5</td>
</tr>
<tr>
<td>Ramesh</td>
<td>K</td>
<td>Narayan</td>
<td>555555555</td>
<td></td>
<td>38000</td>
<td>222222222</td>
<td>5</td>
</tr>
<tr>
<td>Joyce</td>
<td>A</td>
<td>English</td>
<td>666666666</td>
<td></td>
<td>25000</td>
<td>222222222</td>
<td>5</td>
</tr>
<tr>
<td>Alicia</td>
<td>J</td>
<td>Zelaya</td>
<td>777777777</td>
<td></td>
<td>25000</td>
<td>333333333</td>
<td>4</td>
</tr>
<tr>
<td>Jennifer</td>
<td>S</td>
<td>Wallace</td>
<td>333333333</td>
<td></td>
<td>43000</td>
<td>111111111</td>
<td>4</td>
</tr>
<tr>
<td>Ahmad</td>
<td>V</td>
<td>Jabbar</td>
<td>888888888</td>
<td></td>
<td>25000</td>
<td>333333333</td>
<td>4</td>
</tr>
<tr>
<td>James</td>
<td>E</td>
<td>Borg</td>
<td>111111111</td>
<td></td>
<td>55000</td>
<td>NULL</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\Rightarrow
\]

<table>
<thead>
<tr>
<th>DNO</th>
<th>EMPS</th>
<th>AVG(SALARY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>33250</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>31000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>55000</td>
</tr>
</tbody>
</table>

Then, the groups are aggregated.
Query 5

SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE ( SELECT COUNT(*)
        FROM DEPENDENT
        WHERE SSN = ESSN ) >= 2

Query 21

For every project, the project number and name as well as the number of employees working for the project should be retrieved.

SELECT PNUMBER, PNAME, COUNT(*) AS EMPS
FROM PROJECT, WORKS_ON
WHERE PNUMBER = PNO
GROUP BY PNUMBER, PNAME
Grouping with Having

Query 22

For every project with more than two employees working on it, the project number and name as well as the number of employees working for the project should be retrieved.

```
SELECT PNUMBER, PNAME, COUNT(*) AS EMPS
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT(*) > 2
```
Result of Query 22:

First, the relations PROJECT and WORKS_ON are joined using the condition PNO=PNUMBR.

<table>
<thead>
<tr>
<th>PNAME</th>
<th>PNUMBER</th>
<th>. . .</th>
<th>ESSN</th>
<th>PNO</th>
<th>HOURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductX</td>
<td>1</td>
<td>. . .</td>
<td>444444444</td>
<td>1</td>
<td>32.5</td>
</tr>
<tr>
<td>ProductX</td>
<td>1</td>
<td>. . .</td>
<td>666666666</td>
<td>1</td>
<td>20.0</td>
</tr>
<tr>
<td>ProduktY</td>
<td>2</td>
<td>. . .</td>
<td>444444444</td>
<td>2</td>
<td>7.5</td>
</tr>
<tr>
<td>ProduktY</td>
<td>2</td>
<td>. . .</td>
<td>666666666</td>
<td>2</td>
<td>20.0</td>
</tr>
<tr>
<td>ProduktY</td>
<td>2</td>
<td>. . .</td>
<td>222222222</td>
<td>2</td>
<td>10.0</td>
</tr>
<tr>
<td>ProductZ</td>
<td>3</td>
<td>. . .</td>
<td>555555555</td>
<td>3</td>
<td>40.0</td>
</tr>
<tr>
<td>ProductZ</td>
<td>3</td>
<td>. . .</td>
<td>222222222</td>
<td>3</td>
<td>10.0</td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
<td>. . .</td>
<td>222222222</td>
<td>10</td>
<td>10.0</td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
<td>. . .</td>
<td>777777777</td>
<td>10</td>
<td>10.0</td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
<td>. . .</td>
<td>888888888</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
<td>. . .</td>
<td>222222222</td>
<td>20</td>
<td>10.0</td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
<td>. . .</td>
<td>333333333</td>
<td>20</td>
<td>15.0</td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
<td>. . .</td>
<td>111111111</td>
<td>20</td>
<td>NULL</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>. . .</td>
<td>888888888</td>
<td>30</td>
<td>5.0</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>. . .</td>
<td>333333333</td>
<td>30</td>
<td>20.0</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>. . .</td>
<td>777777777</td>
<td>30</td>
<td>30.0</td>
</tr>
</tbody>
</table>
Then the condition “**HAVING COUNT(*) > 2**” is evaluated.

<table>
<thead>
<tr>
<th>PNAME</th>
<th>PNUMBER</th>
<th>. . .</th>
<th>ESSN</th>
<th>PNO</th>
<th>HOURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProduktY</td>
<td>2</td>
<td>. . .</td>
<td>4444444444</td>
<td>2</td>
<td>7.5</td>
</tr>
<tr>
<td>ProduktY</td>
<td>2</td>
<td>. . .</td>
<td>6666666666</td>
<td>2</td>
<td>20.0</td>
</tr>
<tr>
<td>ProduktY</td>
<td>2</td>
<td>. . .</td>
<td>2222222222</td>
<td>2</td>
<td>10.0</td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
<td>. . .</td>
<td>2222222222</td>
<td>10</td>
<td>10.0</td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
<td>. . .</td>
<td>7777777777</td>
<td>10</td>
<td>10.0</td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
<td>. . .</td>
<td>8888888888</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
<td>. . .</td>
<td>2222222222</td>
<td>20</td>
<td>10.0</td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
<td>. . .</td>
<td>3333333333</td>
<td>20</td>
<td>15.0</td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
<td>. . .</td>
<td>1111111111</td>
<td>20</td>
<td>NULL</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>. . .</td>
<td>8888888888</td>
<td>30</td>
<td>5.0</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>. . .</td>
<td>3333333333</td>
<td>30</td>
<td>20.0</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>. . .</td>
<td>7777777777</td>
<td>30</td>
<td>30.0</td>
</tr>
</tbody>
</table>
And finally the groups are aggregated.

<table>
<thead>
<tr>
<th>PNUMBER</th>
<th>PNAME</th>
<th>EMPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ProductY</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>Computerization</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>Reorganization</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>Newbenefits</td>
<td>3</td>
</tr>
</tbody>
</table>
**Rules for SELECT Statements with GROUP BY and HAVING**

1. First, the relations from the **FROM** clause are joined based on the selection and join conditions from the **WHERE** clause.

2. Then, **GROUP BY** groups, and the aggregate functions are applied to the groups; NULL values are ignored during the aggregation.

3. The **HAVING** clause selects individual groups.

4. Finally, the **SELECT** clause returns attribute values that have been used in the grouping (and only such), together with the computed aggregate values.

5. Thus, all non–aggregated attributes from the **SELECT** clause have to appear in the **GROUP BY**.
**Substring Comparisons**

**Query 25**

Find all employees living in Houston, Texas.

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE ADDRESS LIKE '%Houston, TX%'
```

**Query 26**

Find all employees born in the 50ies.

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE BDATE LIKE '_ _ 5 _ _ _ _ _'
```

Alternatively: BDATE LIKE ’__ 5%’
Arithmetic Operations, Order, and Formatting of the Output

Query 27
Show the salaries of the employees working for ProductX after a rise of salary by 10%.

\[
\begin{align*}
\text{SELECT} & \quad \text{FNAME, LNAME, 1.1*SALARY} \\
\text{FROM} & \quad \text{EMPLOYEE, WORKS_ON, PROJECT} \\
\text{WHERE} & \quad \text{SSN = ESSN} \\
\text{AND} & \quad \text{PNO = PNUMBER AND PNAME = 'ProductX'} \\
\text{ORDER BY} & \quad \text{FNAME, LNAME}
\end{align*}
\]

For sorting FNAME descendingly, the \text{ORDER BY} clause has to be replaced by

\[
\begin{align*}
\text{ORDER BY} & \quad \text{FNAME Desc, LNAME Asc.}
\end{align*}
\]

Using \text{SELECT} \quad \text{FNAME, ':', LNAME, ':', 1.1*SALARY} \quad \text{wie could achieve a ':'}–separated output for later import in MS Excel.
If nothing else is stated, then the standard ordering is ascending.

**Query 28**

```
SELECT DNAME, FNAME, LNAME, PNAME
FROM DEPARTMENT, EMPLOYEE,
  WORKS_ON, PROJECT
WHERE DNUMBER = DNO
  AND SSN = ESSN
  AND PNO = PNUMBER
ORDER BY DNAME Desc, LNAME, FNAME
```
2.3 Update Statements in SQL

For *inserting* a single tuple into a relation, *(INSERT)* offers two possibilities:

1. We can list the values of the new tuple in the same order as in the CREATE TABLE statement.

2. We can list the attributes and the corresponding values to be inserted explicitly.

In MySQL, several tuples can be inserted separated by commas in a single INSERT statement.
Example

\begin{verbatim}
INSERT INTO EMPLOYEE
VALUES ('Richard', 'K', 'Marini', '653298653', '1952-12-30',
        '98 Oak Forest, Katy, TX', 'M', 37000, '333333333', 4 )

INSERT INTO EMPLOYEE(FNAME, LNAME, SSN, DNO)
VALUES ('Richard', 'Marini', '653298653', 4),
        ('Robert', 'Hatcher', '980760540', 2)
\end{verbatim}

The check of the \textit{integrity constraints} is system dependent. The following operation is refused, since no value is given for the primary key “SSN”:

\begin{verbatim}
INSERT INTO EMPLOYEE(FNAME, LNAME, DNO)
VALUES ('Robert', 'Hatcher', 2)
\end{verbatim}
Also, a *relation* derived using a **SELECT** statement can be inserted:

```
CREATE TABLE DEPT (  
  DNAME        VARCHAR(15),  
  EMPS         INTEGER,  
  TOTAL_SAL    INTEGER  
);  
```

```
DEPT
DNAME  | EMPS | TOTAL_SAL
-------|------|-----------
Administration | 3    | 93000
Headquarters    | 1    | 55000
Research         | 4    | 133000
```

```
INSERT INTO DEPT  
SELECT DNAME, COUNT(*), SUM(SALARY)  
FROM DEPARTMENT, EMPLOYEE  
WHERE DNUMBER = DNO  
GROUP BY DNAME
```
The **DELETE** operation deletes tuples according to a **WHERE** condition; here, the check of the referential integrity constraints is system dependent.

```
DELETE FROM EMPLOYEE
WHERE LNAME='Brown'
```

```
DELETE FROM EMPLOYEE
WHERE DNO IN ( 
    SELECT DNUMBER 
    FROM DEPARTMENT 
    WHERE DNAME = 'Research' 
)
```

```
DELETE FROM EMPLOYEE
```

The last statement – without **WHERE** clause – deletes all tuples.
The **UPDATE** Statement

- The **WHERE** clause specifies the tuples to be modified.
- The **SET** clause specifies the attributes to be modified and their corresponding new values.

**UPDATE**  PROJECT

**SET**  PLOCATION=’Bellaire’, DNUM=5

**WHERE**  PNUMBER=10
Several tuples can be modified at the same time using a computation rule:

```
UPDATE  EMPLOYEE
SET     SALARY = SALARY * 1.1
WHERE   DNO IN ( 
    SELECT DNUMBER
    FROM    DEPARTMENT
    WHERE   DNAME = 'Research'
)
```

As in programming languages, an attribute name in the right hand side of an assignment represents the old value, while in the left hand side it represents the new value.
2.4 Views in SQL

A View is a virtual table that is derived from other tables (stored base tables or other views). These other tables are called defining tables.

A view has a (virtual) table name, a list of attributes, and its contents is specified using a SELECT statement:

```
CREATE VIEW E_WORKS_ON_P
AS SELECT FNAME, LNAME, PNAME, HOURS
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE SSN=ESSN AND PNO=PNUMBER
```
<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>PNAME</th>
<th>HOURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>ProductX</td>
<td>32.5</td>
</tr>
<tr>
<td>John</td>
<td>Smith</td>
<td>ProductY</td>
<td>7.5</td>
</tr>
<tr>
<td>Ramesh</td>
<td>Narayan</td>
<td>ProductZ</td>
<td>40.0</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
<td>ProductX</td>
<td>20.0</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
<td>ProductY</td>
<td>20.0</td>
</tr>
<tr>
<td>Franklin</td>
<td>Wong</td>
<td>ProductY</td>
<td>10.0</td>
</tr>
<tr>
<td>Franklin</td>
<td>Wong</td>
<td>ProductZ</td>
<td>10.0</td>
</tr>
<tr>
<td>Alicia</td>
<td>Zelaya</td>
<td>Newbenefits</td>
<td>30.0</td>
</tr>
<tr>
<td>Alicia</td>
<td>Zelaya</td>
<td>Computerization</td>
<td>10.0</td>
</tr>
<tr>
<td>Ahmad</td>
<td>Jabbar</td>
<td>Computerization</td>
<td>35.5</td>
</tr>
<tr>
<td>Ahmad</td>
<td>Jabbar</td>
<td>Newbenefits</td>
<td>5.0</td>
</tr>
<tr>
<td>Jennifer</td>
<td>Wallace</td>
<td>Newbenefits</td>
<td>20.0</td>
</tr>
<tr>
<td>Jennifer</td>
<td>Wallace</td>
<td>Reorganization</td>
<td>15.0</td>
</tr>
<tr>
<td>James</td>
<td>Borg</td>
<td>Reorganization</td>
<td>NULL</td>
</tr>
</tbody>
</table>
CREATE VIEW DEPT (DNAME, EMPS, TOTAL_SAL) AS SELECT DNAME, COUNT(*), SUM(SALARY) FROM DEPARTMENT, EMPLOYEE WHERE DNUMBER=DNO GROUP BY DNAME

<table>
<thead>
<tr>
<th>DEPT</th>
<th>DNAME</th>
<th>EMPS</th>
<th>TOTAL_SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Administration</td>
<td>3</td>
<td>93000</td>
</tr>
<tr>
<td></td>
<td>Headquarters</td>
<td>1</td>
<td>55000</td>
</tr>
<tr>
<td></td>
<td>Research</td>
<td>4</td>
<td>133000</td>
</tr>
</tbody>
</table>

For efficiency reasons, a view should be deleted, as soon as it is not needed any more in a session:

DROP VIEW E_WORKS_ON_P

DROP VIEW DEPT
Queries can be posed to a view in the usual way. This supports the repeated posing of similar queries. 

```
SELECT FNAME, LNAME
FROM E_WORKS_ON_P
WHERE PNAME = 'ProjectX'
```

A view is always current, it automatically reflects changes resulting from modifications of the defining tables.

There are two strategies for efficiently implementing views:

1. query modification: maps a query to a view to a query to its defining tables.

2. view materialisation: a view is computed when the first query to it is posed; an incremental method is used for propagating changes in the defining tables to the view.
2.5 Indexes in SQL

An index is a *physical access structure*. It supports the access to tuples using conditions referring to the indexed attributes.

```
CREATE INDEX LNAME_INDEX
ON EMPLOYEE (LNAME);

CREATE INDEX NAMES_INDEX
ON EMPLOYEE (LNAME ASC, FNAME DESC, MINIT);
```

The **CREATE TABLE** statements automatically create an index on the primary key and the secondary keys.
The key word **UNIQUE** after the **CREATE** declares the indexed attributes to be a key.

```
CREATE UNIQUE INDEX NAMES_INDEX
ON EMPLOYEE (LNAME, FNAME, MINIT);
```

A base relation can have at most one “cluster index” – this is used for organising the data records physically on the storage media.

But it can have many other indexes.

```
CREATE INDEX DNO_INDEX
ON EMPLOYEE (DNO)
CLUSTER;
```

The maintenance of an index – reacting to changes to the defining relations – is time consuming; thus, an index should be *deleted* as soon as it is not needed any more.

```
DROP INDEX DNO_INDEX;
```
2.6 Java Database Connectivity (JDBC)

SQL can be used together with conventional procedural programming languages like JAVA, C++, or C; then, these are called host languages.

```java
import java.io.*;
import java.sql.*;
import java.util.*;
import java.text.*;
import java.lang.reflect.*;

class MySQL_Query {

    protected final String dbPassword = "...";
    protected final String dbUser = "seipel";
    protected String jdbcURL = "jdbc:mysql://localhost/company";
```
protected String
    jdbcDriver = "org.gjt.mm.mysql.Driver";

    /* connection to the database */
protected Connection con = null;

public static void main(String args[])
{
    MySQL_Query query = new MySQL_Query();
    query.initConnection();
    query.salary_query();
}

protected void initConnection()
    ...
private void salary_query()
    ...
}
protected void initConnection()
{
    try {
        Class.forName(jdbcDriver).newInstance();
        con = DriverManager.getConnection(
            jdbcURL, dbUser, dbPassword);
    }
    catch (InstantiationException ie) {
        System.out.println(ie + ie.getMessage());
    }
    catch (IllegalAccessException iae) {
        System.out.println(iae + iae.getMessage());
    }
    catch (SQLException sqle) {
        System.out.println(sqle + sqle.getMessage());
    }
    catch (ClassNotFoundException cnfe) {
        System.out.println(cnfe + cnfe.getMessage());
    }
}
private void salary_query()
{
    try {
        String query =
                "SELECT * FROM employee WHERE salary >= 30000";
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            String fname = rs.getString(1);
            String minit = rs.getString(2);
            String lname = rs.getString(3);
            String salary = rs.getString(8);
            System.out.println( "( " + fname + ", " + minit +
                    ", " + lname + ", " + salary + ")" );
        }
    } catch (Exception mye){
        System.out.println(mye.toString());
    }
}