STRUCTURED QUERY LANGUAGE (SQL)

- Standard for relational databases
- Differences between implementations of SQL in commercial DBMSs.
- If the programmer does not use exotic features and both systems adhere to the standard, conversion between code written for two systems is much easier.
- A database application program can contain code to access data in two (or more) different DBMSs.

- We have seen one of the most important formalisms of the relational data model.
- **Relational Algebra** is important for query processing and optimization and gives us an idea of what kind of requests we can specify on a relational database.
- **RA** is beautiful but has the inconvenience that the user must specify the **order** of execution of the operations.
- **SQL** provides a high-level declarative interface, the user has to specify only what the result of the query will be.

- **SQL** contains some features from Relational Algebra and tuple relational calculus (another formalism for the relational data model).
- **SQL** is the standard language for commercial RDBMSs.
- In **SQL** we can create tables, define, query and update relational data, define views, specify security, authorization, integrity constraints
- **SQL** is both a DDL and a DML

Brief SQL history

- ANSI+ISO ➔SQL-86 (alias SQL1)
- Current standard: SQL-92 (alias SQL2) has 3 levels: ***Entry SQL, Intermediate SQL, Full SQL***
- SQL:1999➔SQL3, Object-oriented features, recursive queries, enhanced embedded SQL features, transaction capabilities

Data Definition in SQL

- Table (relation) Row (tuple) Column (attribute)
- SQL2 commands for data definition: ***CREATE, ALTER, DROP***
- ***Schema, Catalog*** concepts in SQL-92:
  - ***SQL schema*** = schema name+authorization identifier+descriptors for schema elements (tables constraints, views, domains) e.g.

    CREATE SCHEMA MOVIES **AUTHORIZATION** IKOTSIRE;
  - ***Catalog*** = set of available schemas+constraints info+authorization info+element descriptors

CREATE TABLE, Data types & Constraints

- Specify a new relation (name & attributes & constraints)
- Each attribute is given a name and a data types plus constraints (if any, e.g. ***NOT NULL***)
- Key, entity integrity, referential integrity constraints are also specified
- **CREATE TABLE** MOVIES.FILM …
- Attribute data types: numeric, string, date, time, timestamp

Schema Evolution Commands

- When a whole schema is not needed → **DROP SCHEMA** command
- Two options: CASCADE, RESTRICT
- **DROP SCHEMA** MOVIES **CASCADE**;
- Delete MOVIES and all its tables, domains etc
- **DROP SCHEMA** MOVIES **RESTRICT**;
- Delete MOVIES only if it has no elements in it.
- When a table is not needed → **DROP TABLE**

- **DROP TABLE** has the CASCADE, RESTRICT options
- *RESTRICT*: the table is deleted only if it is *not referenced* in any constraints
- Change the definition of a table → **ALTER TABLE** command
- **ALTER TABLE** possible actions: add/drop attributes/constraints, change definitions of attributes

- **ALTER TABLE** MOVIES.AWARD **ADD** AWARDNAME VARCHAR(20); *add attribute*
- Values for the new attribute must be provided for each AWARD tuple (**UPDATE** command) otherwise the default value NULL is assigned in all tuples
- **ALTER TABLE** MOVIES.AWARD **DROP** YEAR **CASCADE|RESTRICT**; *drop attribute*
- *RESTRICT* → no views or constraints reference the attr.
- **ALTER TABLE** MOVIES.AWARD **DROP/ADD CONSTRAINT** INT **CASCADE|RESTRICT**; *add/drop constraint* (the constraint must have a name)

BASIC QUERIES IN SQL

- Basic SQL statement for retrieving information from the database: ***SELECT***
- SQL **allows** duplicate elements in the result (as opposed to RA queries) multiset/set
- Basic syntax of the ***SELECT*** command:

```
SELECT  <attribute list>
FROM     <table list>
WHERE    <condition>;
```

Q0: Retrieve the birthdates and addresses of the employees whose last name is Smith.

```
SELECT      BDATE, ADDRESS
FROM        EMPLOYEE
WHERE       LNAME = ‘Smith’;
```

This corresponds to the RA query:

$$\pi_{\text{BDATE,ADDRESS}} (\sigma_{\text{LNAME=‘Smith’}}(\text{EMPLOYEE}))$$

Q1: Retrieve the names and addresses of all employees who work for the Research department.

```
        SELECT      FNAME, LNAME, ADDRESS
        FROM        EMPLOYEE, DEPARTMENT
        WHERE       DNAME = ‘Research’ AND DNUMBER=DNO;
```

JOIN CONDITION: DNUMBER = DNO corresponds to a RA JOIN operation

Q2: For every project located in Stratford retrieve the project number, the controlling dpt, and the manager’s last name and birthdate.

```
        SELECT      PNUMBER, DNUM, LNAME, BDATE
        FROM        PROJECT, EMPLOYEE, DEPARTMENT
        WHERE       DNUM=DNUMBER AND MGRSIN=SIN
AND PLOCATION=‘Stratford’;
```

CORRELATED NESTED QUERIES

Whenever a condition in the WHERE clause of an inner nested query references an attribute of a relation of the FROM clause of the outer query, the two (nested) queries are called **correlated**.

Evaluation Mechanism: for each tuple (or combination of tuples) of the outer query, the inner query is evaluated and the outer query tuple is selected or not, accordingly.

EXISTS

Usage: check whether the result of a correlated (inner) nested query is empty or not

Q16: Retrieve the names of each employee who has a dependent with the same first name and the same sex as the employee.

```
SELECT      E.FNAME, E.LNAME
FROM        EMPLOYEE AS E
WHERE       EXISTS   ( SELECT *
                       FROM DEPENDENT
                       WHERE E.SIN=ESIN AND E.SEX=SEX
                          AND E.FNAME=DEPENDENT_NAME);
```

NOT EXISTS

Q6: Retrieve the names of employees who have no dependents

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       NOT EXISTS ( SELECT *
                         FROM DEPENDENT
                         WHERE SIN=ESIN);
```

Q7: Retrieve the names of managers who have at least 1 dependent

```
SELECT     FNAME, LNAME
FROM       EMPLOYEE
WHERE    EXISTS ( SELECT * FROM DEPENDENT WHERE SIN=ESIN )
      AND EXISTS ( SELECT * FROM DEPARTMENT WHERE SIN=MGRSIN);
```

EXISTS-NOT EXISTS SEMANTICS

EXISTS(Q) is true when there is at least one tuple in the result of query Q.
(the result of query Q is not empty)

NOT EXISTS(Q) is true when there are no tuples in the result of query Q.
(the result of query Q is empty)

EXCEPT (set-theoretic difference)

Q3: Retrieve the name of each employee who works on **all** the projects managed by department number 5.

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       NOT EXISTS (
                  ( SELECT PNUMBER FROM PROJECT  WHERE DNUM=5 )
                   EXCEPT
                   ( SELECT PNO FROM WORKS_ON WHERE SIN=ESIN )
                 );
```

Explicit Sets of Values

We can use an **explicit set of values** instead of an inner nested query in the WHERE-clause of an SQL statement.

This set of values must be delimited by parentheses.

Q17: Retrieve the SIN of all employees who work on project number 1, 2 or 3.

```
SELECT     DISTINCT ESIN
FROM       WORKS_ON
WHERE      PNO IN (1,2,3);
```

Substring Pattern Matching

Q12: Find all employees living in Stratford

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       ADDRESS LIKE ‘%Stratford%’;
```

% matches zero or more characters (* in Linux)

_ matches one single character (? in Linux)

Q12A: Find all employees born in the 1950s

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       BDATE LIKE ‘_ _ 5 _%’;
```

Arithmetic Operators
Comparison Operators

Q13: show a hypothetical 10% raise for all employees working in department number 5

```
SELECT     FNAME, LNAME, 1.1*SALARY AS incrSalary
FROM       EMPLOYEE
WHERE      DNO = 5;
```

Q14: retrieve all employees working in department number 5 whose salary is between $30,000 and $40,000

```
SELECT     FNAME, LNAME
FROM       EMPLOYEE
WHERE      (SALARY BETWEEN 30000 AND 40000) AND
           DNO =5;
```

NESTED QUERIES AND SET/MULTISET COMPARISONS

Q: find all employees who work on the same project and the same number of hours, on some project that employee ‘123456789’ works on

```
SELECT      DISTINCT ESIN
FROM        WORKS_ON
WHERE       (PNO,HOURS) IN
                 (SELECT PNO, HOURS
                  FROM WORKS_ON
                  WHERE SIN=‘123456789’);
```

We can compare tuples of (union-compatible) values, as opposed to single individual values, using parentheses.

NESTED QUERIES AND SET/MULTISET COMPARISONS

Q: find all employees whose salary is greater than the salary of all the employees in department number 5.

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

ALL/ANY SEMANTICS

u > ALL V is true if the value u is greater than all the values in the set (multiset) S
u = ANY V is true if the value u is equal to some value in the set (multiset) S

S is typically specified by a nested query
In some SQL implementations ANY is called SOME

Ordering of Query Results

Q15: retrieve a list of employees and the projects they are working on, ordered by department and within each department ordered ABtically by last name

```
SELECT     DNAME, LNAME, FNAME, PNAME
FROM       EMPLOYEE, DEPARTMENT, WORKS_ON, PROJECT
WHERE      DNUMBER=DNO AND SIN=ESIN AND
           PNO=PNUMBER
ORDER BY   DNAME, LNAME;
```

Default: **ASC** (ascending order)

ORDER BY DNAME **DESC**

NULL values in SQL

We can check whether a value in a tuple is **NULL**.
SQL provides two comparison operators, **IS**, **IS NOT**

Q18: Retrieve the names of all employees who do not have supervisors

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       SUPERSIN IS NULL;
```

SQL considers all null values as being different end thus equality comparison is meaningless.
→In case of a join condition, tuples with null values are not included in the result

Attribute/Relation Aliasing(Renaming)

Using the qualifier **AS** we can rename/alias

- attributes in the SELECT-clause
- relations in the FROM-clause

Q8: For each employee retrieve his/her last name and the last name of his/her immediate supervisor

```
SELECT      E.LNAME AS EMPL_NAME, S.LNAME AS SUPER_NAME
FROM        EMPLOYEE AS E, EMPLOYEE AS S
WHERE       E.SUPERSIN = S.SIN;
```

The new (attribute) names will appear in the query result

Joined Tables

Usage: to be able to specify a table resulting from a join operation, in the FROM-clause of a query

Q1: Retrieve the name and address of all employees working in the "Research" department

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

In RA terms, we separate the join and the project

NATURAL JOIN:

no join condition specified,
an implicit join condition is applied to every pair of attributes with the same name.

Q1: with a **NATURAL JOIN**

```
SELECT    FNAME, LNAME, ADDRESS
FROM      ( EMPLOYEE NATURAL JOIN
           (DEPARTMENT AS DEPT(DNAME,DNO,MSIN,MSDATE)) )
WHERE     DNAME= “Research”;
```

We renamed the attributes of the relation ***DEPARTMENT*** *to match the attribute* ***DNO*** *of the relation* ***EMPLOYEE***

Q2: with **2 NESTED JOINs**

Q2: For every project located in “Stratford”,
list the project number, the controlling department number
and the department manager’s last name, address and birthdate

```
SELECT      PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM        ( (PROJECT JOIN DEPARTMENT ON DNUM=DNUMBER )
              JOIN EMPLOYEE ON MGRSIN=SIN )
WHERE       PLOCATION= “Stratford”;
```

AGGREGATE FUNCTIONS & GROUPING

Built-in functions: MAX, MIN, COUNT, SUM, AVG

The COUNT function returns the number of tuples in the result of a query

The functions MAX, MIN, SUM, AVG are applied
to a set (or multiset) of numeric values.

These functions can be used in the **SELECT**-clause or the **HAVING**-clause of a query.

MIN, MAX can be used with attributes whose domains have a total order

Use aggregate functions to retrieve summary values

Q19 Compute the sum of the salaries of all employees, the maximum salary, the minimum salary and the average salary

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

Q20 Compute the sum of the salaries of all employees of the “Research” department as well as the max, min, average salary in this department

```
SELECT     SUM(SALARY), MAX(SALARY),
           MIN(SALARY), AVG(SALARY)
FROM       EMPLOYEE, DEPARTMENT
WHERE      DNO=DNUMBER AND DNAME= “Research”;
```

Q21 Compute the total number of employees in the company

```
SELECT      COUNT(*)
FROM        EMPLOYEE;
```

Q22 Compute the number of employees in the “Research” department

```
SELECT      COUNT(*)
FROM        EMPLOYEE, DEPARTMENT
WHERE       DNO=DNUMBER AND DNAME= “Research”;
```

The * refers to rows, #tuples in the result of the query

The **COUNT** function can be used on columns too:

Q23 Count the number of all distinct salary values

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

COUNT(SALARY) is equivalent to: **COUNT**(*)

Use aggregate functions to select particular tuples

Use a correlated nested query with the aggregate function in the WHERE-clause of an outer query

<u>Q5</u> Retrieve the names of all employees who have two or more dependents

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       (  SELECT COUNT(*)
               FROM DEPENDENT
               WHERE SIN=ESIN   ) >=  2;
```

The correlated inner query counts the number of dependents of each employee

Apply aggregate functions to subgroups of tuples

clause **GROUP BY**

Work with subgroups of tuples sharing one (or more) common attribute value(s)

❑ Group the tuples according to attribute(s)

❑ Apply the aggregate function to each group separately

Q24 For each department retrieve the dept. number, the number of employees and their average salary

```
SELECT     DNO, COUNT(*), AVG(SALARY)
FROM       EMPLOYEE
GROUP BY   DNO;
```

Q25 For each project retrieve the project number the project name and the number of employees who work on that project

```
SELECT      PNUMBER, PNAME, COUNT(*)
FROM        PROJECT, WORKS_ON
WHERE       PNUMBER=PNO
GROUP BY    PNUMBER, PNAME;
```

First we perform the join of the two relations
Then we perform the grouping

Apply aggregate functions to groups that satisfy certain conditions

clause **HAVING**

Q26 For each project **with more than two employees working on it**, retrieve the project number the project name and the number of employees who work on that project

```
SELECT          PNUMBER, PNAME, COUNT(*)
FROM            PROJECT, WORKS_ON
WHERE           PNUMBER=PNO
GROUP BY        PNUMBER, PNAME
HAVING          COUNT(*) > 2;
```

Q27 For each project retrieve the project number the project name and the number of employees from department 5 who work on that project

```
SELECT      PNUMBER, PNAME, COUNT(*)
FROM        PROJECT, WORKS_ON, EMPLOYEE
WHERE       PNUMBER=PNO AND SIN=ESIN AND DNO=5
GROUP BY          PNUMBER, PNAME;
```

Q28 For each department with more than five employees working on it, retrieve the department number and the number of its employees making more than $40,000

```
SELECT      DNUMBER, COUNT(*)
FROM        DEPARTMENT, EMPLOYEE
WHERE       DNUMBER=DNO AND SALARY > 40000
GROUP BY          DNAME
HAVING            COUNT(*) > 5;
```

WRONG

```
SELECT      DNUMBER, COUNT(*)
FROM        DEPARTMENT, EMPLOYEE
WHERE       DNUMBER=DNO AND SALARY > 40000 AND
            DNO IN   (  SELECT       DNO
                        FROM         EMPLOYEE
                        GROUP BY     DNO
                        HAVING       COUNT(*) > 5   )
GROUP BY DNUMBER;
```

Summary of SQL queries

- Six clauses.
- Only SELECT and FROM are mandatory
- General Form:

```
SELECT <attr/fct list>
FROM     <table list>
WHERE  <condition(s)>
GROUP BY <grouping attr>
 HAVING   <group condition>
 ORDER BY  <attr list>;
```

- ***Conceptual Evaluation*** of a query:
  - apply the FROM clause (identify the tables involved)
  - Apply WHERE, GROUP BY, HAVING
  - Apply ORDER BY to sort the query result
- For a SELECT-FROM-WHERE query:

  for each combination of tuples (FROM clause) evaluate the WHERE clause. If the result is true, retrieve the attributes specified in the SELECT clause in the result table

INSERT (1)

- Add a single tuple to a relation
- Specify the relation name and a list of values
- Values listed in the same order as in the CREATE TABLE command
- **INSERT INTO EMPLOYEE**

  VALUES (‘John’,’L’,’Smith’,’950120230’,…);

INSERT (2)

- Specify explicit attribute names corresponding to the values provided in the INSERT command
- Specify the relation name and a list of values
- Values listed in the same order as in the CREATE TABLE command
- **INSERT INTO** EMPLOYEE(LNAME,DNO,SIN) **VALUES** (‘Smith’,4,’ ’950120230’);
- Attributes not specified → set to NULL/DEFAULT

DELETE

- **DELETE**: remove tuples (0, 1, more) from a table
- Uses a WHERE clause to select the tuples
- Missing WHERE clause → deletion of all tuples
-
  ```
  DELETE FROM EMPLOYEE
  WHERE           LNAME = ‘Smith’;
  ```
-
  ```
  DELETE FROM EMPLOYEE
  WHERE  DNO IN  ( SELECT DNUMBER
                   FROM   DEPARTMENT
                   WHERE  DNAME = ‘Research’);
  ```

UPDATE

- **UPDATE**: modify attribute values of one or more selected tuples
- Uses a WHERE clause to select the tuples to be modified
- Primary Key Update → Propagated actions (Ref. Integrity)
-
```
UPDATE PROJECT
SET    PLOCATION='London', DNUM = 5;
WHERE  PNUMBER = 10;
```
-
```
UPDATE  EMPLOYEE
SET     SALARY = SALARY*1.1
WHERE   DNO IN ( SELECT  DNUMBER
                 FROM    DEPARTMENT
                 WHERE   DNAME = 'Research');
```

Give all employees of the Research Dept. a 10% raise

VIEWS (1)

- **View** == single table *derived* from other tables
- The other tables can be existing tables or other views and they are called the *defining tables*
- Views are *intermediate* tables that do not exist physcically → *virtual tables*
- Views can be queried just like other tables
- Views are tables than need to be referenced frequently, even though they don’t exist physically

VIEWS (2)

- Example: *instead of* issuing frequently queries to retrieve the employee name and the project names the employee works on (this requires a **JOIN**) we *define a view* as the result of this join and query the view (this requires a **single-table** retrieval)
- The tables **EMPLOYEE**, **WORKS_ON** and **PROJECT** are the *defining tables* of the view