The SQL Standard

- SQL – Structured Query Language
  An international standard (ANSI, ISO) that specifies how
  - a relational schema is created
  - data is inserted / updated in the relations
  - data is queried
  - transactions are started and stopped
  - programs access data in the relations
  - and a host of other things are done
- Every relational database management system (RDBMS) is required to support / implement the SQL standard.
  - RDBMS vendors may give additional features
  - Downside of using vendor-specific features - portability

History of SQL

SEQUEL
- developed by IBM in early 70's
- relational query language as part of System-R project at IBM San Jose Research Lab.
- the earliest version of SQL

SQL evolution
- SQL- 86/89
- SQL- 92 - SQL2
- SQL- 99/03 - SQL3
  (includes object relational features)

And the evolution continues .....

Disclaimer: This module covers only important principles of SQL

Components of SQL Standard(1/2)

- *Data Definition Language* (DDL)
  Specifies constructs for schema definition, relation definition, integrity constraints, views and schema modification.
- *Data Manipulation Language* (DML)
  Specifies constructs for inserting, updating and querying the data in the relational instances (or tables).
- *Embedded SQL and Dynamic* SQL
  Specifies how SQL commands can be embedded in a high-level host language such as C, C++ or Java for programmatic access to the data.

Components of SQL Standard(2/2)

- *Transaction Control*
  Specifies how transactions can be started / stopped, how a set of concurrently executing transactions can be managed.
- *Authorization*
  Specifies how to restrict a user / set of users to access only certain parts of data, perform only certain types of queries etc.

Data Definition in SQL

Defining the schema of a relation

create table *r* (attributeDefinition-1, attributeDefinition-2,…,
attributeDefinition-n, [integrityConstraints-1],
[integrityConstraints-2],…,[integrityConstraints-m])

name of the relation

Attribute Definition –

attribute-name domain-type [NOT NULL] [DEFAULT v]

E.g.:

create table *example1* (A char(6) not null default "000000",
B int, C char(1) default "F");

Domain Types in SQL-92 (1/2)

- *Numeric data types*
  - integers of various sizes – INT, SMALLINT
  - real numbers of various precision – REAL, FLOAT, DOUBLE PRECISION
  - formatted numbers – DECIMAL (i, j) or NUMERIC (i, j)
    i – total number of digits (precision)
    j – number of digits after the decimal point (scale)
- *Character string data types*
  - fixed length – CHAR(n) – n: no. of characters
  - varying length – VARCHAR(n) – n: max.no. of characters
- *Bit string data types*
  - fixed length – BIT(n)
  - varying length – BIT VARYING(n)

Domain Types in SQL-92 (2/2)

- *Date data type*

  DATE type has 10 position format – YYYY-MM-DD

- *Time data type*

  TIME type has 8 position format – HH : MM : SS

- *Others*

  There are several more data types whose details are available in SQL reference books

Specifying Integrity Constraints in SQL

Also called Table Constraints
Included in the definition of a table

Key constraints
PRIMARY KEY (A_1,A_2,…,A_k)
specifies that {A_1,A_2,…,A_k} is the primary key of the table

UNIQUE (B_1,B_2,…,B_k)
specifies that {B_1,B_2,…,B_k} is a candidate key for the table

There can be more than one UNIQUE constraint but only one PRIMARY KEY constraint for a table.

Specifying Referential Integrity Constraints

FOREIGN KEY (A_1) REFERENCES r_2 (B_1)

- specifies that attribute A_1 of the table being defined, say r_1, is a *foreign key* referring to attribute B_1 of table r_2
- recall that this means:
  each value of column A_1 is either null or is one of the values appearing in column B_1 of r_2

Specifying What to Do if RIC Violation Occurs

RIC violation

- can occur if a referenced tuple is deleted or modified
- action can be specified for each case using qualifiers
  ON DELETE or ON UPDATE

Actions

- three possibilities can be specified
  SET NULL, SET DEFAULT, CASCADE
- these are actions to be taken on the referencing tuple
- SET NULL – foreign key attribute value to be set null
- SET DEFAULT – foreign key attribute value to be set to its default value
- CASCADE – delete the referencing tuple if the referenced tuple is deleted or update the FK attribute if the referenced tuple is updated

Table Definition Example

```
create table students (
       rollNo char(8) not null,
       name varchar(15) not null,
       degree char(5),
       year smallint,
       sex char not null,
       deptNo smallint,
       advisor char(6),
       primary key(rollNo),
       foreign key(deptNo) references
                              department(deptId)
              on delete set null on update cascade,
       foreign key(advisor) references
                              professor(empId)
              on delete set null on update cascade
 );
```


Modifying a Defined Schema

ALTER TABLE command can be used to modify a schema

Adding a new attribute

ALTER table student ADD address varchar(30);

Deleting an attribute

- need to specify what needs to be done about views or constraints that refer to the attribute being dropped
- two possibilities
  CASCADE – delete the views/constraints also
  RESTRICT – do not delete the attributes if there are some views/constraints that refer to it.
- ALTER TABLE student DROP degree RESTRICT
  Similarly, an entire table definition can be deleted

Data Manipulation in SQL

Basic query syntax

select $A_1,A_2,\ldots,A_m$ ← a set of attributes from relations $R_1,\ldots,R_p$ that are required in the output table.

from $R_1,R_2,\ldots,R_p$ ← the set of tables that contain the relevant tuples to answer the query.

where θ ← a boolean predicate that specifies when a combined tuple of $R_1,\ldots,R_p$ contributes to the output.

Equivalent to:

$$\pi_{A_1,A_2,\ldots A_n}(\sigma_\theta(R_1\times R_2\times\ldots..\times R_p))$$

Assuming that each attribute name appears exactly once in the table.

Meaning of the Basic Query Block

- The *cross product M* of the tables in the from clause would be considered.
  Tuples in *M* that satisfy the condition θ are *selected*.
  For each such tuple, values for the attributes $A_1,A_2,\ldots,A_m$ (mentioned in the select clause) are *projected*.
- This is a conceptual description
  - in practice more efficient methods are employed for evaluation.
- The word *select* in SQL should not be confused with select operation of relational algebra.

SQL Query Result

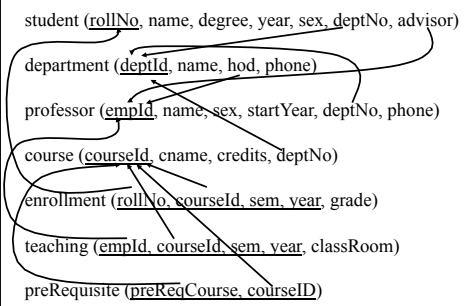
The result of any SQL query

- a table with *select* clause attributes as column names.
- duplicate rows may be present.
  - differs from the definition of a relation.
- duplicate rows can be eliminated by specifying DISTINCT keyword in the *select* clause, if necessary.

```
SELECT DISTINCT name
FROM student WHERE …
```

- duplicate rows are essential while computing aggregate functions (average, sum etc).
- removing duplicate rows involves additional effort and is done only when necessary.

Example Relational Scheme with RICs shown



Example Queries Involving a Single Table

Get the rollNo, name of all women students in the dept no. 5.

```
select rollNo, name
from student
where sex = 'F' and deptNo = 5;
```

Get the employee Id, name and phone number of professors in the CS dept (deptNo = 3) who have joined after 1999.

```
select empId, name, phone
from professor
where deptNo = 3 and startYear > 1999;
```


Examples Involving Two or More Relations (1/2)

Get the rollNo, name of students in the CSE dept (deptNo = 3)along with their advisor's name and phone number.

```
select rollNo, s.name, f.name as advisorName,
       phone as advisorPhone
from student as s, professor as f
where s.advisor = f.empId and
      s.deptNo = 3;
```

attribute renaming in the output

table aliases are used to disambiguate the common attributes

table aliases are required if an attribute name appears in more than one table.
Also when *same* relation appears twice in the from clause.

Examples Involving Two or More Relations (2/2)

Get the names, employee ID's, phone numbers of professors in CSE dept who joined before 1995.

```
select empId, f.name, f.phone
from professor as f, department as d
where f.deptNo = d.deptId and
      d.name = 'CSE' and
      f.startYear < 1995
```


Nested Queries or Sub-queries

While dealing with certain complex queries

- beneficial to specify part of the computation/requirement as a separate query and make use of its result to formulate the main query.
- such queries – nested / sub-queries.

Using sub-queries

- makes the main query easy to understand / formulate
- sometimes makes it more efficient also
  - sub query result can be computed once and used many times.
    - not the case with all sub-queries.

Nested Query Example

Get the rollNo, name of students who have a lady professor as their advisor.

IN Operator: One of the ways of making use of the subquery result

Subquery computes the empId's of lady professors

```
select s.rollNo, s.name
from student s
where s.advisor IN
        (select empId
         from professor
         where sex = 'F');
```

NOT IN can be used in the above query to get details of students who do not have a lady professor as their advisor.

Set Comparison Operators

SQL supports several operators to deal with subquery results or in general with collection of tuples.

Combination of { =, <, ≤, ≥, >, < > } with keywords { ANY, ALL } can be used as set comparison operators.

Get the empId, name of the senior-most Professor(s):

```
select p.empId, p.name
from professors p
where p.startYear <= ALL ( select distinct startYear
                           from professor );
```


Semantics of Set Comparison Operators

op is one of <, ≤, >, ≥, =, < >

- *v op* ANY *S*
  true if for some member *x* of *S*, *v* op *x* is true
  false if for no member *x* of *S*, *v* op *x* is true

S is a subquery

- *v op* ALL *S*
  true if for every member *x* of *S*, *v* op *x* is true
  false if for some member *x* of *S*, *v* op *x* is not true

- IN is equivalent to = ANY
  NOT IN is equivalent to
  < > ALL

- *v* is normally a single attribute, but while using IN or NOT IN it can be a tuple of attributes

Correlated and Uncorrelated Nested Queries

If the nested query result is independent of the current tuple being examined in the outer query,
nested query is called *uncorrelated,*
otherwise, nested query is called *correlated.*

Uncorrelated nested query
- nested query needs to be computed only once.

Correlated nested query
- nested query needs to be re-computed for each row examined in the outer query.

Example of a Correlated Subquery

Get the roll number and name of students whose gender is same as their advisor's.

```
select s.rollNo, s.name
from student s
where s.sex = ALL ( select f.sex
                    from professor f
                    where f.empId = s.advisor );
```


The *EXISTS* Operator

Using *EXISTS*, we can check if a subquery result is non-empty

EXISTS(*S*) is *true* if *S* has at least one tuple / member
is *false* if *S* contain no tuples

Get the employee Id and name of professors who advise at least one women student.

a correlated subquery

```
select f.empId, f.name
from professors f
where EXISTS ( select s.rollNo
               from student s
               where s.advisor = f.empId and
                     s.sex = 'F' );
```

SQL does not have an operator for universal quantification.

The *NOT EXISTS* Operator

Obtain the department Id and name of departments that do not offer any 4 credit courses.

a correlated subquery

```
select d.deptId, d.name
from department d
where NOT EXISTS ( select courseId
                   from course c
                   where c.deptNo = d.deptId and
                         c.credits = 4 );
```

Queries with *existentially* quantified predicates can be easily specified using *EXISTS* operator.

Queries with *universally* quantified predicates can only be specified after translating them to use *existential* quantifiers.

Example Involving Universal Quantifier

Determine the students who are enrolled for every course taught by Prof Ramanujam. Assume that Prof Ramanujam teaches at least one course.

As SQL does not have universal quantifier, we will rewrite the query this way:

Determine the student(s) who are such that there ***does not*** exist a course taught by Prof Ramanujam which is ***not*** enrolled by the student.

Same query expressed in TRC

Determine the students who are enrolled for ***every*** course taught by Prof Ramanujam. Assume that Prof Ramanujam teaches at least one course.

```
1.  {s.rollNo | student (s) ^
2.    (∀c)(course (c) ^
3.          (∃t)(∃p)( ( teaching(t) ^ professor(p) ^
4.              t.courseId = c.courseId ^
5.              p.name = "Ramanujam" ^
6.              p.empId = t.empId ) →
7.                (∃e) (enrollment(e) ^
8.                      e.courseId = c.courseId ^
9.                      e.rollNo = s.rollNo ^
10.                     e.sem = t.sem ^
11.                     e.year = t.year
12.                     ) )
13.       )
14.  }
```


Query expressed in SQL

```
select s.rollNo, s.name
from student s
where not exists ( select * from course c, professor  p
                   where p.name = "Ramanujam" and
                     exists (select * from teaching t1
                        where t1.courseId = c.courseId and
                              t1.empId = p.empId )
                   and not exists
                   ( select * from teaching t2, enrolment e
                     where t2.empId = p.empId and
                     t2.courseId = c.courseId and
                          t2.courseId = e.courseId and
                          t2.sem = e.sem and t2.year = e.year and
                          e.rollNo = s.rollNo)
                 );
```

c is taught by Prof Ramanujam

s has not enrolled for c when it was taught by Prof Ramanujam

Another Example Involving the Universal Quantifier

Determine the students who have obtained either S or A grade in all the pre-requisite courses of the course CS7890. It is known that CS7890 has at least one pre-requisite.

```
select s.rollNo, s.name
from student s
where NOT EXISTS (select *
                  from preRequisite p
                  where p.courseId = "CS7890" and
                   NOT EXISTS
                  (select *
                   from enrollment e
                   where e.courseId = p.preReqcourse
                      and e.rollNo = s.rollNo and
                      (e.grade = "S" or e.grade = "A")
                  );
```


Missing *where* Clause

If the *where* clause in an SQL query is not specified, it is treated as - the where condition is true for all tuple combinations.

- Essentially no filtering is done on the cross product of from clause tables.

Get the name and contact phone of all Departments.

```
select name, phone
from department
```


Union, Intersection and Difference Operations

- In SQL, using operators *UNION, INTERSECT* and *EXCEPT*, one can perform set *union, intersection* and *difference* respectively.
- Results of these operators are ***sets*** – i.e duplicates are automatically removed.
- Operands need to be union compatible and also have *same* attribute names in the *same* order.

Example using UNION

Obtain the roll numbers of students who are enrolled for either CS2300 or CS2320 in 2019 odd semester.

```
(SELECT rollNo
    FROM enrollment
    WHERE courseId = 'CS2300' and
            sem = 'odd' and year = '2019' ) UNION
   (SELECT rollNo
    FROM enrollment
    WHERE courseId = 'CS2320' and
           sem = 'odd' and year = '2019' );
```

Equivalent to:

```
   (SELECT rollNo
    FROM enrollment
    WHERE (courseId = 'CS2300' or courseID = 'CS2320')
           and sem = 'odd' and year = '2019' )
```


Example using INTERSECTION

Obtain the roll numbers of students who are enrolled for both CS2300 and CS2320 in 2019 odd semester.

```
      (select rollNo
      from enrollment
      where courseId = 'CS2300' and
            sem = 'odd' and year = '2019' )

      INTERSECT

      (select rollNo
      from enrollment
      where courseId = 'CS2320' and
            sem = 'odd' and year = '2019';
```


Example using EXCEPT

Obtain the roll numbers of students who are not enrolled for CS2300 course in 2019 odd semester.

```
      (SELECT rollNo
      FROM enrollment
      WHERE sem = 'odd' and year = '2019')
        EXCEPT
            (SELECT rollNo
            FROM enrollment
            WHERE courseId = 'CS2300' and
            sem = 'odd' and year = '2019');
```