Structured Query Language
Relational Query Language: SQL

- Supports simple, yet powerful querying of data.
  - Precise semantics for relational queries.
  - DML (Data Manipulation Language)
  - DDL (Data Definition Language)

- SQL developed by IBM (system R) in the 1970s

- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision)
  - SQL-99 (major extensions, triggers, recursive queries)
SQL Data Types

- **Character strings**
  - `CHAR(n), VARCHAR(n)`: fixed and variable-length strings

- **Bits**
  - `BOOLEAN` – values TRUE, FALSE, UNKNOWN
  - `BIT(n)`

- **Numerical:**
  - `INTEGER (INT)`
  - Floating point: `FLOAT` (or `REAL`), `DOUBLE PRECISION`
  - Fixed precision: `DECIMAL(n,d)`
    - `1234.56` is of type `DECIMAL(6,2)`, precision 6, scale 2

- **DATE and TIME**
Creating Relations in SQL

CREATE TABLE Students
    (sid CHAR(20),
     name CHAR(20),
     login CHAR(10),
     age INTEGER,
     gpa REAL);

CREATE TABLE Enrolled
    (sid CHAR(20),
     cid CHAR(20),
     grade CHAR(2));
Destroying and Altering Relations

**DROP TABLE** Students;

- Deletes relation *Students*, including schema information *and* all the tuples

**ALTER TABLE** Students

  **ADD** firstYear INTEGER;

- Add new column to schema
- Every tuple is extended with **NULL** value in added field
- Default value may be specified instead of NULL
### Structure of SQL SELECT Query

<table>
<thead>
<tr>
<th>SELECT</th>
<th>[DISTINCT] target-list</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>relation-list</td>
</tr>
<tr>
<td>WHERE</td>
<td>qualification</td>
</tr>
</tbody>
</table>

- **relation-list** = list of relation names
  - possibly with a *range-variable* after each name
- **target-list** = list of attributes of relations in *relation-list*
- **qualification** = conditions \( \text{Attr op const or Attr}_1 \text{ op Attr}_2 \)
  - \( \text{op} \) is one of \( <, >, =, \geq, \leq, <>, \), or string operators
  - Expressions connected using *AND*, *OR* and *NOT*
- **DISTINCT** = optional, eliminates duplicates
  - By default duplicates are *NOT* eliminated!
Conceptual Evaluation Strategy

- Semantics of SQL query
  1. Compute the cross-product of *relation-list*
  2. Discard resulting tuples if they fail *qualifications*
  3. Delete attributes that are not in *target-list*
  4. If **DISTINCT** is specified, eliminate duplicate rows

- This strategy is least efficient way to compute a query!
  - Optimizer finds efficient strategies to compute *the same result*
## Example Schema

### Sailors

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

### Boats

<table>
<thead>
<tr>
<th>bid</th>
<th>name</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>clipper</td>
<td>green</td>
</tr>
</tbody>
</table>

### Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Conceptual Evaluation Example

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid = 103
```

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>(sid)</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
A Note on Range Variables

- Really needed only if the same relation appears twice in the FROM clause (SELECT … FROM Sailors S1, Sailors S2)

```
SELECT  S.sname
FROM     Sailors S, Reserves R
WHERE    S.sid=R.sid AND R.bid=103
```

It is good style, however, to use range variables always!

Instead of …

```
SELECT  sname
FROM     Sailors, Reserves
WHERE    Sailors.sid=Reserves.sid AND bid=103
```
Duplicate Tuples and DISTINCT

- Would adding `DISTINCT` to this query make a difference?
- What is the effect of replacing `S.sname` by `S.sid` in the `SELECT` clause?
- Would adding `DISTINCT` to this variant of the query make a difference?
Expressions and Strings

“Find rating and number of years to retirement for sailors whose names begin with ‘d’, end with ‘n’ and contain at least three characters”

```sql
SELECT  S.rating, 60 - S.age AS Yr_to_retire
FROM    Sailors S
WHERE   S.sname LIKE ‘d_%n’
```

AS allows to (re)name fields in result.

LIKE is used for string matching

_ stands for any one character

% stands for 0 or more arbitrary characters
Expressions and Strings - Example

SELECT  S.rating, 60 - S.age AS Yr_to_retire
FROM   Sailors S
WHERE  S.sname LIKE 'd_%n'

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rating</th>
<th>Yr_to_retire</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>15</td>
</tr>
</tbody>
</table>
Set Operations

- **UNION**
  - compute the union of any two *union-compatible* sets of tuples

- **INTERSECT**
  - compute the intersection of any two *union-compatible* sets of tuples

- **EXCEPT or MINUS**
  - Set difference of any two *union-compatible* sets of tuples

- Duplicates eliminated by default!
  - UNION ALL, INTERSECT ALL, EXCEPT ALL retain duplicates
  - Contrast with non-set SQL operations
Insert single tuple

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES ('53688', 'Smith', 'smith@ee', 18, 3.2);
```

Delete all tuples satisfying condition

```
DELETE
FROM Students S
WHERE S.name = 'Smith';
```
Data Modifications: Inserts

INSERT INTO Table (attr1, attr2, ...) VALUES (val1, val2, ...);

- Values and attribute domains must match
- Attributes not specified will be assigned value NULL

- Variation: insert tuples returned by SELECT

  INSERT INTO Table (attr1, attr2, ...) SELECT col1, col2, ... FROM ...
  [WHERE ...
  GROUP BY ...
  HAVING ... ];
Data Modifications: Updates

- No new tuples created
- Attribute values of existing tuples modified
  
  ```sql
  UPDATE Table
  SET  attr1=expression1, attr2=expression2 [...] 
  WHERE condition;
  ```

- Values and attribute domains must match
- It is possible to use subqueries:

  ```sql
  UPDATE Table
  SET  attr1= (SELECT value1
                FROM ...
                WHERE ... ) 
  WHERE condition;
  ```
**Integrity Constraints (ICs)**

- **IC**: condition that must hold for *any* instance of the database; e.g., *domain constraints*
  - Specified when schema is defined.
  - Checked when relations are modified.

- A *legal* instance satisfies all specified ICs
  - It is the DBMS’s role to enforce IC

- **ICs we study**
  - Primary key constraints
  - Foreign key constraints
  - Referential integrity
Primary and Candidate Keys in SQL

- **Primary keys** specified by keyword **PRIMARY KEY**
- **Candidate keys** specified by keyword **UNIQUE**

**Distinctions between the two:**
- Any attribute in the primary key is **NOT** allowed to have **NULL** values
- Primary key attributes may have special roles in the DBMS internals (although from the logical point of view is same as unique)

**Declaration**
- In-line with the respective attribute
  - Only if one-attribute key!
- Or as separate constraint line
### Keys in SQL - Examples

#### Schema and Instance

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Smith</td>
<td>20</td>
</tr>
<tr>
<td>53650</td>
<td>Jones</td>
<td>25</td>
</tr>
<tr>
<td>53681</td>
<td>Adams</td>
<td>22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cname</th>
<th>room</th>
</tr>
</thead>
<tbody>
<tr>
<td>114</td>
<td>Calculus</td>
<td>M123</td>
</tr>
<tr>
<td>115</td>
<td>Databases</td>
<td>M234</td>
</tr>
</tbody>
</table>

#### Enrolled

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>114</td>
<td>A</td>
</tr>
<tr>
<td>53650</td>
<td>115</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>115</td>
<td>B</td>
</tr>
</tbody>
</table>
Keys in SQL - Examples

“For a given student and course, there is a single grade.”

CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid))

“Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade.”

CREATE TABLE Enrolled
(sid CHAR(20) PRIMARY KEY,
cid CHAR(20),
grade CHAR(2),
UNIQUE (cid, grade) )
Foreign Keys, Referential Integrity

- **Foreign key**
  - Set of fields in relation A that refer to a tuple in relation B
  - Must correspond to primary key of relation B (or UNIQUE)
- Not necessary for field names in A and B to be the same!!!
  - FOREIGN KEY (attr1) REFERENCES B (attr2)
- E.g. `sid` in Enrolled is a foreign key referring to Students:
  - Enrolled(\(sid: \text{string}\), \(cid: \text{string}\), \(grade: \text{string}\))

- **Referential integrity** is achieved by enforcing all foreign keys
  - no “dangling references”
Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses

CREATE TABLE Enrolled
    (sid CHAR(20), cid CHAR(20), grade CHAR(2),
     PRIMARY KEY (sid,cid),
     FOREIGN KEY (sid) REFERENCES Students )

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>114</td>
<td>A</td>
</tr>
<tr>
<td>53650</td>
<td>115</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>115</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Smith</td>
<td>20</td>
</tr>
<tr>
<td>53650</td>
<td>Jones</td>
<td>25</td>
</tr>
<tr>
<td>53681</td>
<td>Adams</td>
<td>22</td>
</tr>
</tbody>
</table>