**SQL**

- Tables are SQL entities that correspond to relations.
- Language for describing database schemas and manipulating tables.
- SQL:92 is the current standard but will be replaced by SQL:1999.
- Declarative - Statement specifies goal, not how it is to be achieved.
- Simplifies applications programs.
- But programmers should have an idea of strategies used by DBMS so they can design tables, queries, so that DBMS can evaluate queries efficiently.

**RELATIONAL SCHEMA**

- Relations names
- Attributes names and domains
- Constraints
- Default values

---

**CREATING TABLES**

- **PRIMARY/CANDIDATE KEY:**
  
  **Course:**

  ```
  CREATE TABLE Course(
  CrsCode CHAR(6) NOT NULL,
  CrsName CHAR(20) NOT NULL,
  DeptId CHAR(4) NOT NULL,
  Descr CHAR(100),
  PRIMARY KEY (CrsCode),
  UNIQUE (DeptId, CrsName)); -- Candidate key
  ```

- **NULL:**
  - Not all data must be known when a row is inserted. (e.g. Grade may be missing from Enrolled).
  - A column may not be applicable for a particular row (e.g. MaidenName if row describes a male).
  - Primary key constrained by NOT NULL.
- **DEFAULT:**
  A default value is assigned if attribute value in a row is not specified.

**Student:**

```sql
CREATE TABLE Student(
  Id INTEGER NOT NULL,
  Name CHAR(20) NOT NULL,
  Address CHAR(50),
  status CHAR(10) DEFAULT 'freshman',
  PRIMARY KEY (id));
```

**SEMANTIC CONSTRAINTS:**
- Used for application dependent conditions.
- Each row in the table must satisfy the condition.
- Empty table always satisfies all CHECK constraints.
- Test made at insertion.

**Enrolled:**

```sql
CREATE TABLE Enrolled(
  StudId INTEGER NOT NULL,
  CrsCode CHAR(6) NOT NULL,
  Semester CHAR(6) NOT NULL,
  Grade CHAR(1),
  PRIMARY KEY (CrsCode, StudId, Semester),
  CHECK (Grade in ('A', 'B', 'C', 'D', 'F')),
  CHECK (StudId > 0 AND StudId < 1000000000));
```

**Teaching:**

```sql
CREATE TABLE Teaching(
  CrsCode CHAR(6) NOT NULL,
  Semester CHAR(6) NOT NULL,
  ProfId INTEGER,
  PRIMARY KEY (CrsCode, Semester),
  FOREIGN KEY (CrsCode) references Course,
  FOREIGN KEY (ProfId) references Professor(Id));
```

**FOREIGN KEY**

- Example:
  **Teaching:**

```sql
CREATE TABLE Teaching(
  CrsCode CHAR(6) NOT NULL,
  Semester CHAR(6) NOT NULL,
  ProfId INTEGER,
  PRIMARY KEY (CrsCode, Semester),
  FOREIGN KEY (CrsCode) references Course,
  FOREIGN KEY (ProfId) references Professor(Id));
```

- Problem 1:
  Circularity in Foreign key constraint - Creation of table A requires existence of the table B and vice versa.

- Solution 1:
  ```sql
  CREATE TABLE A (...) -- no foreign key
  CREATE TABLE B (...) -- foreign keys included
  ALTER TABLE A
  ADD CONSTRAINT cons
  FOREIGN KEY (...) REFERENCES B(…)
  ```
**HANDLING EVENTS**

- Constraints enable DBMS to recognize a bad state and reject the statement or transaction that creates it.

- Mechanism to allow a user to specify an arbitrary situation and an action to be taken if that situation occurs.

- SQL:92 provides a mechanism for handling foreign key violations.

**EXAMPLE**

```
CREATE TABLE Teaching(
    ProfId INTEGER,
    CrsCode CHAR(6) NOT NULL,
    Semester CHAR(6) NOT NULL,
    PRIMARY KEY(CrsCode, Semester),
    FOREIGN KEY(CrsCode)
    REFERENCES Course(CrsCode),
    ON DELETE SET NULL
    ON UPDATE CASCADE,
    FOREIGN KEY(ProfId)
    REFERENCES Professor(Id)
    ON DELETE NO ACTION
    ON UPDATE CASCADE);
```
**DATATYPES**

Borland Interbase

```sql
<datatype> = {SMALLINT | INTEGER | FLOAT
| DOUBLE PRECISION}[<array_dim>]
| {DATE | TIME | TIMESTAMP}[<array_dim>]
| {DECIMAL | NUMERIC} [{precision [, scale]}][<array_dim>]
| {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [{int}] [<array_dim>]
| {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [{VARYING} [{int}]} [<array_dim>]
| BLOB [{SUB_TYPE {int | subtype_name}}
| SEGMENT SIZE int] [CHARACTER SET charname]
| BLOB [{seglen [, subtype]}]<array_dim> = [{x:y} [, {x:y}]]
```

**MODIFYING THE SCHEMA**

- **Delete a table:**
  ```sql```
  DROP TABLE Teaching
  ```

- **Addition of a column to a table:**
  ```sql```
  ALTER TABLE Student
  ADD Gpa INTEGER DEFAULT 0
  ```

- **Addition of a constraint:**
  ```sql```
  ALTER TABLE Student
  ADD CONSTRAINT GpaRange
  CHECK (Gpa >= 0 AND Gpa <= 4)
  ```

- **Removing a constraint:**
  ```sql```
  ALTER TABLE Student
  DROP CONSTRAINT GpaRange
  ```

**ASSERTION**

- Not in Borland Interbase.
- Part of the schema.
- Specification of an inter-relational constraint.
- An assertion applies to the entire database.

- **Example:**
  ```sql```
  CREATE TABLE Employee(
  Id INTEGER NOT NULL,
  Name CHAR(20),
  Salary INTEGER,
  PRIMARY KEY (Id));
  ```

  CREATE ASSERTION DontFireEveryone
  CHECK (0 < SELECT COUNT(*) FROM Employee)
  ```

**TRIGGERS**

- General mechanism in SQL:1999 for handling events.
- **Example:**
  See in Borland Interbase SQL Reference guide.
### Access control

- Databases contain sensitive data.
- Access has to be limited to:
  - Users have to be identified (authentication)
  - Users can access only to appropriate data (authorization)
- Controlling authorization using GRANT (SQL:92):
  
  ```sql
  GRANT access_list ON table TO user_list
  access_mode E {SELECT, INSERT, DELETE, UPDATE, REFERENCE}
  GRANT SELECT ON Enrolled TO joe
  GRANT UPDATE (Grade) ON Enrolled TO prof.smith
  ```

### Tuples in Tables

- **Insertion:**
  ```sql
  INSERT INTO Student(Id,Name,Address,Status) VALUES (999999999,'Bill','432 pine', 'senior')
  ```
- **Update:**
  ```sql
  UPDATE Student
  SET Status = 'sophomore'
  WHERE Id = 11111111
  ```
- **Deletion:**
  ```sql
  DELETE FROM Student
  WHERE ID=11111111
  ```

### Query Sub-language of SQL

#### SELECT

- **Syntax:**
  ```sql
  SELECT <attributes list>
  FROM <table(s)>
  WHERE <condition>
  - FROM: indicates initial table(s)
  - WHERE: indicates the rows to retain (SELECTION, σ)
  - SELECT: indicates which columns to extract from retained rows (PROJECTION, π)
  
  The result is a table.
  ```

### Example

- **SELECT Name**
  ```sql
  FROM Student
  WHERE Id > 4999
  ```
  Equivalent to: \( \pi_{\text{Name}}(\text{Student}, \text{Id} > 4999) \)

- **Student:**
<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John</td>
<td>123 Main</td>
<td>freshman</td>
</tr>
<tr>
<td>5522</td>
<td>Mary</td>
<td>77 Pine</td>
<td>senior</td>
</tr>
<tr>
<td>9876</td>
<td>Bill</td>
<td>83 Oak</td>
<td>junior</td>
</tr>
</tbody>
</table>

- **Result:**
  | Name | Mary  | Bill  |
### EXAMPLES

- SELECT Id, Name
  FROM Student
  WHERE Status = 'senior'

- SELECT *
  FROM Student
  WHERE Status = 'senior' AND Id > 6000

---

**Example**

- Student(Id, Name, Address, Status)
- Enrolled(StudId, CrsCode, Semester, Grade)
- Names, courses taken and grades of the seniors.

  SELECT Name, CrsCode, Grade
  FROM Student, Enrolled
  WHERE StudId=Id AND Status = 'senior'

  Equivalent to:

  \[ \pi_{Name, CrsCode, Grade}(\sigma_{Status = 'senior'}(\text{Student}) \bowtie_{StudId=Id \text{ } Enrolled}) \]

### Correspondence between SQL and relational algebra

- SELECT a1, b1
  FROM T1, T2
  WHERE a2 = b2

  T1
  a1  a2  a3
  A 1    xxy
  B 17   rst

  T2
  b1  b2
  A 1    3.2  17
  B 17   4.8  17

  FROM T1, T2 yields:

  a1  a2  a3  b1  b2
  A 1    xxy  3.2  17
  A 1    xxy  4.8  17
  B 17   rst  3.2  17
  B 17   rst  4.8  17

- WHERE a2 = b2 yields:

  a1  a2  a3  b1  b2
  B 17   rst  3.2  17
  B 17   rst  4.8  17

- SELECT a1, b1 yields:

  a1  b1
  B 4.8

- SELECT C.CrsName
  FROM Course C, Teaching T
  WHERE C.CrsCode = T.CrsCode AND T.Semester = 'S2000'

  is equivalent to:

  \[ \pi_{\text{CrsName}}(\text{C.CrsCode} = \text{T.CrsCode} \text{ } \text{T.Semester} = 'S2000') \times \\
  \pi_{\text{CrsName, DeptId, CrsName, Desct}}(\text{Course}(\text{C.CrsCode, DeptId, CrsName, Desct})) \times \\
  \pi_{\text{ProfId, T.CrsCode, Semester}}(\text{Teaching}(\text{ProfId, T.CrsCode, Semester})) \]

- Relational algebra expressions are procedural. Which of the two equivalent expressions is more easily evaluated?
• Part of the external schema.

• A (virtual) table constructed from the (actual) tables of the conceptual schema:
  – can be accessed in queries
  – not present physically but computed/constructed when accessed.

```
CREATE VIEW CoursesTaken(StudId, CrsCode, Semester)
AS
SELECT T.StudId, T.CrsCode, T.Semester
FROM Enrolled T

SELECT *
FROM CoursesTaken
```

• Advantage:
  – Customization: users need not see full complexity of database.
  – Users can have access to view and not the tables of the database (External schema). Example:
    GRANT SELECT ON CoursesTaken TO joe