SQL

RELATIONAL SCHEMA

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

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.

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:

```
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:

```
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));
```

EXAMPLES

- **Student:**

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

- **Professor:**

```
CREATE TABLE Professor(
Id INTEGER NOT NULL,
Name CHAR(20),
DeptID CHAR(4),
PRIMARY KEY(Id));
```

- **Course:**

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

- **Enrolled:**

```
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:**

```
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:

```
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:

```
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.

Handling foreign key violation

- Let $A(a_1, ..., a_n)$ and $B(\underline{b_1}, ..., b_m)$.

  a_n in A is a foreign key referencing b_1 in B.

- **Insertion into A:**

  The insertion into A is rejected if no row exists in B containing the foreign key of the inserted row.

- **Deletion from B (DELETE):**

  – NO ACTION (default): The deletion from B is rejected if there is a row in A referencing the row to delete.

  – SET NULL: Set value of foreign key in the referencing row(s) to null.

  – SET DEFAULT: Set value of foreign key in the referencing row(s) to default value.

  – CASCADE: Delete referencing rows.

- **Update candidate key in B (UPDATE):**

  – NO ACTION (default): The update is rejected if there is a row in A referencing the row to update.

  – SET NULL: Set value of foreign key in the referencing row(s) to null.

  – SET DEFAULT: Set value of foreign key in the referencing row(s) to default value.

  – CASCADE: Propagate the new value to the foreign key.

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

```
<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>] [CHARACTER SET charname]

| {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:**

```
DROP TABLE Teaching
```

- **Addition of a column to a table:**

```
ALTER TABLE Student
ADD Gpa INTEGER DEFAULT 0
```

- **Addition of a constraint:**

```
ALTER TABLE Student
ADD CONSTRAINT GpaRange
CHECK (Gpa >= 0 AND Gpa <= 4)
```

- **Removing a constraint:**

```
ALTER TABLE Student
DROP CONSTRAINT GpaRange
```

TRIGGERS

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

ASSERTION

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

```
CREATE TABLE Employee(
Id INTEGER NOT NULL,
Name CHAR(20),
Salary INTEGER,
PRIMARY KEY (Id));


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

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):

```
GRANT access_list ON table TO user_list
```

access_mode $\in$ {SELECT, INSERT, DELETE, UPDATE, REFERENCE}

```
GRANT SELECT ON Enrolled TO joe

GRANT UPDATE (Grade) ON Enrolled TO prof_smith
```

TUPLES IN TABLES

- **Insertion:**

  INSERT INTO Student(Id,Name,Address,Status) VALUES (999999999,'Bill','432 pine', 'senior')

- **Update:**

```
UPDATE Student
SET Status ='sophomore'
WHERE Id = 11111111
```

- **Deletion:**

```
DELETE FROM Student
WHERE ID=11111111
```

QUERY SUB-LANGUAGE OF SQL

SELECT

- Syntax:

```
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
  FROM Student
  WHERE Id > 4999
  ```

  Equivalent to: $\pi_{Name}(Student, Id > 4999)$

- **Student:**

Id	Name	Address	Status
1234	John	123 Main	freshman
5522	Mary	77 Pine	senior
9876	Bill	83 Oak	junior

- **Result:**

Name
Mary
Bill
  ```

EXAMPLES

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

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

JOIN

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

- **T1**

a1	a2	a3
A	1	xxy
B	17	rst

T2

b1	b2
3.2	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	3.2
B	4.8

Example

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

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

  Equivalent to:

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

Correspondence between SQL and relational algebra

- ```
  SELECT C.CrsName
  FROM Course C, Teaching T
  WHERE C.CrsCode = T.CrsCode AND T.Semester ='S2000'
  ```
- is equivalent to:
- $\pi_{CrsName}\sigma_{CCrsCode=TCrsCode\ \ AND\ \ Semester='S2000'}$
  $(Course[CCrsCode, DeptId, CrsName, Descr])X$
  $Teaching[ProfId, TCrsCode, Semester])$
- Relational algebra expressions are procedural. Which of the two equivalent expressions is more easily evaluated?

VIEWS

- 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
  ```