4b. Structured Query Language 3
SQL as Data Definition Language

- Creates the **Students** relation. The type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

  ```sql
  CREATE TABLE Students
  (sid: CHAR(20),
   name: CHAR(20),
   login: CHAR(10),
   age: INT,
   gpa: REAL)
  ```

- As another example, the **Enrolled** table holds information about courses that students take.

  ```sql
  CREATE TABLE Enrolled
  (sid: CHAR(20),
   cid: CHAR(20),
   grade: CHAR(2))
  ```
Integrity Constraints (IC)

IC guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.

- Integrity constraints are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know name is not a key, but the assertion that sid is a key is given to us.
Domain Constraints

- The **check** clause in SQL-92 permits domains to be restricted.
- Use **check** clause to ensure that an hourly-wage domain allows only values greater than a specified value.

```
create domain hourly-wage numeric(5,2)
constraint value-test check (value>=4.00)
```

- The domain hourly-wage is declared to be a decimal number with 5 digits, 2 of which are after the decimal point.
- The domain has a constraint that ensures that the hourly-wage is greater than 4.00.
- **constraint** value-test is optional; useful to indicate which constraint an update violated.
Primary and Candidate Keys in SQL

• Possibly many candidate keys (specified using UNIQUE), one of which is chosen as the primary key.

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

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

• Used carelessly, an IC can prevent the storage of database instances that arise in practice!
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>Carnatic101</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>Reggae203</td>
<td>B</td>
</tr>
<tr>
<td>53650</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>History105</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>
Enforcing Referential Integrity

• Consider Students and Enrolled; \( sid \) in Enrolled is a foreign key that references Students.
• What should be done if an Enrolled tuple with a non-existent student id is inserted? (Reject it!)
• What should be done if a Students tuple is deleted?
  – Also delete all Enrolled tuples that refer to it (cascading deletion).
  – Disallow deletion of a Student tuple that is referred to by an Enrolled tuple.
  – Set \( sid \) in Enrolled tuples that refer to it to a default \( sid \).
  – Set \( sid \) in Enrolled tuples that refer to it to a special value \( \text{null} \) (not applicable in this example because \( sid \) is part of the primary key).
• If primary key of Students tuple is updated, you must also update the classes taken by the student
**Referential Integrity in SQL/92**

- SQL/92 supports all 4 options on deletes and updates.
  - Default is NO ACTION (*delete is rejected*)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```sql
CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid)
REFERENCES Students
ON DELETE CASCADE
ON UPDATE CASCADE)
```
Participation Constraints in SQL

We can capture total participation constraints using **NOT NULL**.

```sql
CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    HKID CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (HKID) REFERENCES Employees,
    ON DELETE NO ACTION)
```
Destroying and Altering Relations

DROP TABLE Students
Destroys the relation Students. The schema information \textit{and} the tuples are deleted.

ALTER TABLE Students
ADD COLUMN firstYear: integer

The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a \textit{null} value in the new field.
Provide a mechanism to hide certain data from the view of certain users. To create a view we use the command:

```
create view view-name as <query expression>
where: <query expression> is any legal SQL query
```

EXAMPLE: Create a view from loan(loan-number, branch-name, amount) that hides the amount.
```
create view branch-loan as
  select branch-name, loan-number
  from loan
```

QUERY: Find all loans in the Perryridge branch
```
select loan-number
from branch-loan
where branch-name = “Perryridge”
```

A user who has access to the view, but not the loan table, cannot see the amount.
General Constraints

Useful when more general ICs than keys are involved.
Are created in the definition of table - checked whenever there is an update within the table

CREATE TABLE Loan
( loan-number INTEGER,
  amount INTEGER,
  branch-name CHAR(20),
PRIMARY KEY (loan-number ),
FOREIGN KEY (branch-name) REFERENCES Branch,
  ON DELETE CASCADE
CHECK ( amount >= 1 AND amount <= 10000)
CHECK ( branch-name <> "Choi Hung")
)
Assertions

• An assertion is a complex constraint that the database must always satisfy.

• An assertion in SQL-92 takes the form
  
  `create assertion <assertion-name> check <predicate>`

• Difference from general constraints:
  – A constraint is associated with a single table and checked when there is an update on this specific table
  – An assertion may be associated with several tables, and is checked every time there is an update anywhere.

• Assertion testing may introduce a significant amount of overhead; hence assertions should be used with great care.

• Any predicate allowed in SQL can be used.
• The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

```
create assertion sum-constraint check
(not exists (select * from branch
   where (select sum(amount) from loan
     where loan.branch-name=branch.branch-name)
   >=
   (select sum(amount) from account
     where loan.number-name=branch.branch-name) ))
```

• Note that the assertion refers to multiple tables. Therefore it cannot be included as a constraint in the definition of loan or amount.