Creating & Dropping Tables

Basic syntax:

```sql
CREATE TABLE tableName (attribute1 type1, attribute2 type2, ..., attributeN typeN);
```

```sql
DROP TABLE tableName;
```

Types available:
- INT or INTEGER
- REAL or FLOAT
- CHAR(n), VARCHAR(n)
- DATE, TIME

Example:

```sql
CREATE TABLE Student (SID INTEGER, name CHAR(30), age INTEGER, GPA FLOAT);
CREATE TABLE Take (SID INTEGER, CID CHAR(10));
CREATE TABLE Course (CID CHAR(10), title VARCHAR(100));
```

Keys

Recall: a set of attributes $K$ is a key for a relation $R$ if

1. In no instance of $R$ will two different tuples agree on all attributes of $K$; i.e., $K$ is a “tuple identifier”
2. No proper subset of $K$ satisfies (1); i.e., $K$ is minimal

Declaring Keys

SQL allows multiple keys to be declared for one table:
- At most one PRIMARY KEY per table
- Any number of UNIQUE keys per table
Two places to declare keys in `CREATE TABLE`:
- After an attribute’s type, if the attribute is a key by itself
- As a separate element (essential if key has more than one attribute)

Example:

```sql
CREATE TABLE Student (SID INTEGER PRIMARY KEY,
    name CHAR(30),
    age INTEGER,
    GPA FLOAT);
CREATE TABLE Take (SID INTEGER,
    CID CHAR(10),
    PRIMARY KEY(SID, CID));
CREATE TABLE Course (CID CHAR(10) PRIMARY KEY,
    title VARCHAR(100) UNIQUE);
```

Why declare keys?
- They are “integrity constraints” enforced by the DBMS
- They tell the DBMS to expect frequent lookups using key values

**Keys vs. FD’s in SQL**

Recall in the pure relational model (where every relation is duplicate-free):
- \( K \) is a tuple identifier for \( R \iff K \rightarrow \text{attrs}(R) \)
- \( K \) is a tuple identifier for \( R \iff \{K\}^+ = \text{attrs}(R) \)

In SQL (where a table may contain duplicate tuples):
- If \( K \) is a tuple identifier for \( R \), then \( R \) must be duplicate-free
- \( K \rightarrow \text{attrs}(R) \) and \( \{K\}^+ = \text{attrs}(R) \) may still hold when \( R \) contains duplicates

One can have
- \( K \) is a tuple identifier \( \neq K \rightarrow \text{attrs}(R) \) or \( \{K\}^+ = \text{attrs}(R) \)

Example:

**Views**

A **view** is like a virtual table:
- It is defined by a **view definition query** which describes how to compute the view contents
- DBMS stores the view definition instead of the view contents
- It can be used in queries just like a regular table
Creating & Dropping Views

Syntax:
CREATE VIEW viewName AS viewDefinitionQuery;
DROP VIEW viewName;
Example: StudentRoster view

Example: CS145Roster view

Using Views in Queries

Semantics:
SELECT...FROM..., viewName, ...WHERE...; ≡
SELECT...FROM..., (viewDefinitionQuery) viewName, ...WHERE...;
Example: find the SID of a CS145 student named Bart
→ No more joins!

→ DBMS typically rewrites the query to make it more efficient to evaluate

Why use views?
- To hide some data from the users
- To make certain queries easier or more natural to express
→ Real database applications use tons and tons of views

Modifying Views

- Does not seem to make sense since views are virtual
- But does make sense if that is how user views the database
→ Modify the base tables such that the modification would appear to have been accomplished on the view
Example: DELETE FROM StudentRoster WHERE SID = 123;
Sometimes it is not possible
Example:
CREATE VIEW HighGPAStrudent AS
    SELECT * FROM Student WHERE GPA > 3.7;
INSERT INTO HighGPAStrudent
    VALUES(888, 'Nelson', 10, 2.50);

Sometimes there are too many possibilities
Example:
CREATE VIEW AvgGPA AS
    SELECT AVG(GPA) AS GPA FROM Student;
UPDATE AvgGPA
    SET GPA = 2.5;

Precise conditions for modifiable views are very complicated
SQL2 uses conservative conditions: views must be defined as single-
table SELECT with simple WHERE, no aggregates, no subqueries, etc.

Indexes

An index on attribute $R.A$
- Creates auxiliary persistent data structure
- Can dramatically speed up accesses of the form:
  - $R.A = value$
  - $R.A > value$ (sometimes; depending the type of index)

An index can be built on a combination of multiple attributes as well
Data structures for indexes: sorted lookup tables, hash tables, search
trees, etc. (CS245 and CS346)

Example:
SELECT * FROM Student WHERE name = 'Bart';
Without index on Student.name: because we store tables as flat col-
lections of unordered tuples, we must scan all Student tuples
With index: go “directly” to tuples with name = 'Bart'

Example:
SELECT * FROM Student, Take
    WHERE Student.SID = Take.SID;
Use index on either Student.SID or Take.SID to speed up join

Jun Yang 4 CS145 Spring 1999
Creating & Dropping Indexes

Syntax:
CREATE INDEX  indexName ON  tableName(attr_1, ..., attr_n);
DROP INDEX indexName;

- If CREATE is followed by UNIQUE, DBMS will also enforce that 
  \{attr_1, ..., attr_n\} is a key of tableName
- Choosing which indexes to create is a difficult design issue:
  \[\leadsto\] Depends on the expected query/update load and size of tables