SQL Data Definition and Data Manipulation Languages (DDL and DML)

Note: This handout introduces both the ANSI SQL synatax for the SQL DDL and DML commands, as well as discusses the MySQL extensions to the syntax that are of importance to the course. As a general note, MySQL has a very rich syntax for the statements discussed in this handout, with many advanced features specifiable in addition to the standard ANSI SQL syntax. We only cover the features that are of immediate interest to us.

Data Definition Language.

Creating a Relation

`CREATE TABLE` *Name* (
attribute-declarations
constraint-declarations
)

Attribute declarations:

AttName AttType [default *expression*] [*ColConstraints*]

Constraints

Column constraints:

`[NOT] NULL :` Not null constraint.

`PRIMARY KEY:` Primary key constraint (when the primary key consists of exactly one attribute, otherwise, use constraint declaration).

`UNIQUE:` Key constraint (when the key consists of exactly one attribute, otherwise, use constraint declaration).

`REFERENCES <Table>[(<AttName>)] [ON DELETE CASCADE]:` Foreign key constraint (when the foreign key consists of exactly one attribute, otherwise, use constraint declaration). `ON DELETE CASCADE` specifies that all rows containing a no longer existing value for must be deleted.

`AUTO_INCREMENT:` the values in the column (integer type) are incremented automatically as new tuples are added to the table.

Constraint declarations:

`[constraint <ConstName>] PRIMARY KEY (<AttNames>):` Primary key constraint. Use when the primary key includes multiple attributes.

`[constraint <ConstName>] UNIQUE (<AttNames>):` Key constraint. Use when the key includes multiple attributes.

`[constraint <ConstName>] FOREIGN KEY (<AttNames>) REFERENCES <Table> [(<AttNames>)]:` Foreign key constraint. Use when the foreign key involves multiple attributes.

All column constraints except for not null constraint can only be used if the appropriate constraint (e.g., primary key) is associated with exactly one attribute. (i.e., if your primary key is two attributes, use the constraint declaration, rather than column constraint).

Types

Integer	`INTEGER` or `INT`
	`SMALLINT`
	`TINYINT`
	`MEDIUMINT`
	`BIGINT`
Real	`FLOAT` or `REAL`
	`DOUBLE`
Fixed Point	`DECIMAL`(n, d)
	n - number of digits
	d - number of decimals
	`NUMERIC`(n, d) (Oracle)
Strings	`CHAR`(n)
	n - length of string, max=255
	`VARCHAR`(n),
	n - length of string
Bit Strings	`BIT`(n)
Dates	`DATE`
	`TIME`
	`TIMESTAMP`
	`DATETIME`
	`YEAR`

Examples

```
CREATE TABLE Books (
    LibCode    INT,
    ISBN       CHAR(20),
    Title      CHAR(80),
    Authors    CHAR(60),
    Year       INT,
    Publisher  CHAR(20),
    PurchPrice REAL,
    TakeHome   BOOLEAN,
    PRIMARY KEY (LibCode),
    UNIQUE  (ISBN)
    );

CREATE TABLE Employees (
    SSN  INT PRIMARY KEY,
    Name CHAR(30) NOT NULL,
    Department  INT REFERENCES Departments,
    Salary  FLOAT  NOT NULL
    Position CHAR(30) DEFAULT 'Not Specified',
    StartYear INT CHECK(StartYear > 1992)
     );

CREATE TABLE Departments (
    DeptID INT PRIMARY KEY AUTO_INCREMENT,
    Name CHAR(30) UNIQUE,
    Head INT,
    FOREIGN KEY(Head) REFERENCES Employees
    );
```

Deleting a Table

`DROP TABLE` *Name* [RESTRICT | CASCADE]

Example:

```
DROP TABLE Books;
```

```
DROP TABLE Departments CASCADE;
```

In the latter case, all referential integrity constraints (foreign keys) are **dropped** from their respective tables, after `Departments` table is deleted.

```
DROP TABLE Employees RESTRICT;
```

The use of `RESTRICT` in the `DROP TABLE` command directs the DBMS server to drop the table **only** if doing so does not affect the constraints in other tables. Otherwise, the table is not deleted.

Modifying a Table

- Adding an attribute

`ALTER TABLE` *Name*
`ADD [COLUMN] (` [*AttName Type* [`FIRST` | `AFTER` *AttName*]]+)

Examples:

```
ALTER TABLE  Books
       ADD (Genre CHAR(10),
            NumPages INT);

ALTER TABLE  Employees
       ADD COLUMN (TransferredFrom INT AFTER Department);
```

The `FIRST` and `AFTER` *AttName* modifiers specify the position of the new column in the table. The default position (when both modifiers are omitted) is at the end of the table.

- Deleting an attribute

  `ALTER TABLE` *Name*
  `DROP [COLUMN]` *AttName*+

  Example:

  ```
  ALTER TABLE  Books
         DROP Year;
  ```

- Modifying an attribute

  `ALTER TABLE` *Name*
  `MODIFY` [*AttName Type* [`FIRST` | `AFTER` *AttName*]]+

  Example:

  ```
  ALTER TABLE  Books
         MODIFY Genre VARCHAR2(30);
  ```

- Renaming a table (MySQL only)

  `ALTER TABLE` *Name*
  `RENAME [TO | AS]` *NewName*;

- Adding a contraint

  `ALTER TABLE` *Name*
  `ADD [CONSTRAINT [`*ConstraintId*`]]` *ConstraintSpec*

  Here are some of the contstraint specifications:

Primary Key	`PRIMARY KEY` (*AttName*+)
Candidate Key	`UNIQUE [KEY]` (*AttName*+)
Foreign Key	`FOREIGN KEY`(*AttName*+) `REFERENCES` *Table*(*AttName*+)

  Examples:

  ```
  ALTER TABLE
     ADD PRIMARY KEY(ISBN);

  ALTER TABLE Books
     ADD CONSTRAINT Books_key1 UNIQUE(Title, Author, Publisher, Year);

  ALTER TABLE Books
     ADD FOREIGN KEY(Author) REFERENCES Writers(Name);
  ```

- Removing a constraint.

  To remove a primary key constraint:

  `ALTER TABLE` *Name*
  `DROP PRIMARY KEY`

  To remove a `UNIQUE` constaint:

  `ALTER TABLE` *Name*
  `DROP KEY` *ConstraintId*

  To remove a foreign key constraint:

  `ALTER TABLE` *Name*
  `DROP FOREIGN KEY` *ConstraintId*

- Temporarily disable/enable keys. Sometimes we want to remove constraints on a for a short period of time (for example during a complex insert operation, where referential integrity is difficult to maintain). In these situations, it is often convenient to disable all constraints on the table, perform all necessary operations (ensuring that referential integrity is restored at the end) and re-enable the constraints. The two commands to do it are:

  To disable constraints:

  `ALTER TABLE` *Name*
  `DISABLE KEYS`

  To enable constraints:

  `ALTER TABLE` *Name*
  `ENABLE KEYS`

  Note: the `DISABLE` command suspends the enforcement of constraints but does not remove them. This way, there is no need to add constraints back to the table later.

Note: `ALTER TABLE` command can be used for a wide range of other changes to the database (manipulation of constraints, for example). These are covered later.

Data Manipulation Language

Inserting a Tuple

`INSERT INTO` *TableName(AttNames)*
`VALUES(`*values* `)[,(`*values* `)]*`

`values` — comma-separated list of values. The number of values must match the number attribute names in *AttNames*, and the types must be compatible.

`INSERT INTO` *TableName*
`VALUES(`*values* `)[,(`*values* `)]*`

Values for all attributes must be given and in the order in which attributes were defined in `CREATE TABLE` command.

In both cases, **in MySQL** (but not in ANSI SQL), a single `INSERT INTO` command can insert multiple tuples.

Examples:

```
INSERT INTO Books(LibCode,Title,Year)
VALUES (12349, 'Database Management Systems', 2000);

INSERT INTO Books
VALUES (15923, '1-56592-000-7','Lex \& Yacc',
        'J. Levine, T. Mason, D. Brown', 1990,
         'O'Reily', 29.95, True);

INSERT INTO Departments(
VALUES ('Sales', 3), ('HR',11),
       ('IT', 76);
```

(Note that in the latter example, `DeptID` is an auto incrementing attribute and does not need to be inserted into the table.)

Deleting Tuples

`DELETE FROM` *TableName*
`[WHERE` *Expression*`]`;

Expression identifies the properties of tuples to be removed from the table.

Examples:

```
DELETE FROM Books;

DELETE FROM Books
WHERE LibCode = 12349;

DELETE FROM Books
WHERE PurchPrice > 100.00 AND Year < 1950;
```

Updating Tuples

`UPDATE` *TableName*
`SET` *Assignments*
`WHERE` *Expression*;

Expression identifies tuples to be updated. *Assignments* specifies modifications.

Examples:

```
UPDATE Books
SET Year = 2003
WHERE Year > 2003;

UPDATE Books
SET Year = Year - 1,
    PurchPrice = PurchPrice *1.05;
WHERE Year > 2000;

UPDATE Books
SET TakeHome = True;
```