Fundamentals of Database Systems

Chapter7: Basic SQL

Overview

- **We learned that:**

1. A DBMS will allow us to create and manage databases.
2. We can query the database (retrieve data from tables in the database).
3. We can update the database (change, add, or delete data from tables in the database).

Overview

- **Structured Query Language (SQL):** a programming language used to query and manipulate data.
- DBMSs typically have a Graphical User Interface (GUI)
- ***It is still important to know how to write code to retrieve and manipulate data***
- In the class, we'll go over the important SQL concepts. You'll learn more about SQL in the lab.

Datatypes in SQL

- Every attribute must have a type

- ***Datatypes include:***

1. **Char:** fixed length ‘n’ number of characters. ***Ex:*** Char(2).
2. **Varchar:** variable-length ‘n’ number of characters
3. **Int:** Numerical data. Cannot be a fraction
4. **Float:** Numerical data that can be a fraction
5. **Boolean:** True or False
6. **Date:** ***Ex:*** 2018-01-09
7. **Time:** ***Ex:*** 08:08:57

- Other types also exist

SQL Constraints

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Uniquely identifies a row/record in another table
- CHECK - Ensures that all values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column when no value is specified

Constraint (example 1: NOT NULL)

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

SQL PRIMARY KEY on CREATE TABLE

Constraint (example 2: Primary Key)

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

SQL FOREIGN KEY on CREATE TABLE

Constraint (example 3: Foreign Key)

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

SQL CHECK on CREATE TABLE

Constraint (example 4: Check)

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

SQL DEFAULT Constraint

Constraint (example 5: Default)

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT GETDATE()
);
```

Constraint (example 6: Unique)

```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

Syntax for MySQL

AUTO INCREMENT Field

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

Types of SQL Commands

- ***We will go over these types of commands:***

1. **Create Table**
2. **Querying Relations**
3. **Basic Joins**
4. **Insert**
5. **Update**
6. **Delete**
7. **Aggregate Functions**

- Other types also exist

1. Creating Relations using SQL

- **STATEMENT:**

```
CREATE TABLE table
(
Name_of_attribute        datatype        Null or not Null,
....
....
Name_of_attribute        datatype        Null or not Null,
Constraints
)
```

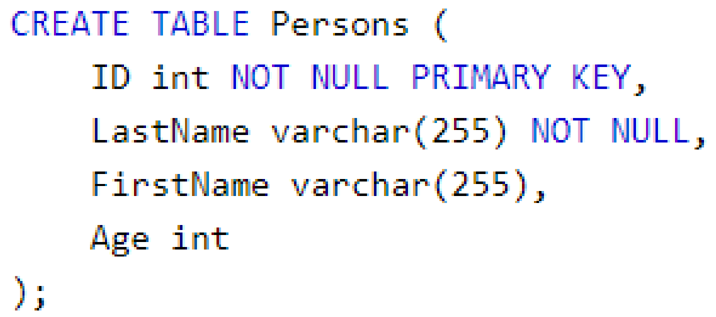
1. Creating Relations using SQL

- **STATEMENT:**

```
CREATE TABLE Student
(
Name            varchar (45)        Not Null,
Student_ID      varchar(10)         Not Null,
Birthdate       date                Null,

Primary Key(Student_ID)
)
```

- ***You can also specify the PK when you declare the variables***



```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

1. Creating Relations using SQL

- **STATEMENT:**

```
CREATE TABLE Students_Grades
(
Student_ID        varchar (10)                          Not Null,
Section_ID        varchar (6)                           Not Null,
Grade             varchar (2)                           Null,
Primary Key (Student_ID, Section_ID),
Foreign Key (Student_ID) REFERENCES Student (Student_ID),
Foreign Key (Section_ID) REFERENCES Section (Section _ID)
)
```

Create TABLE (EXAMPLE)

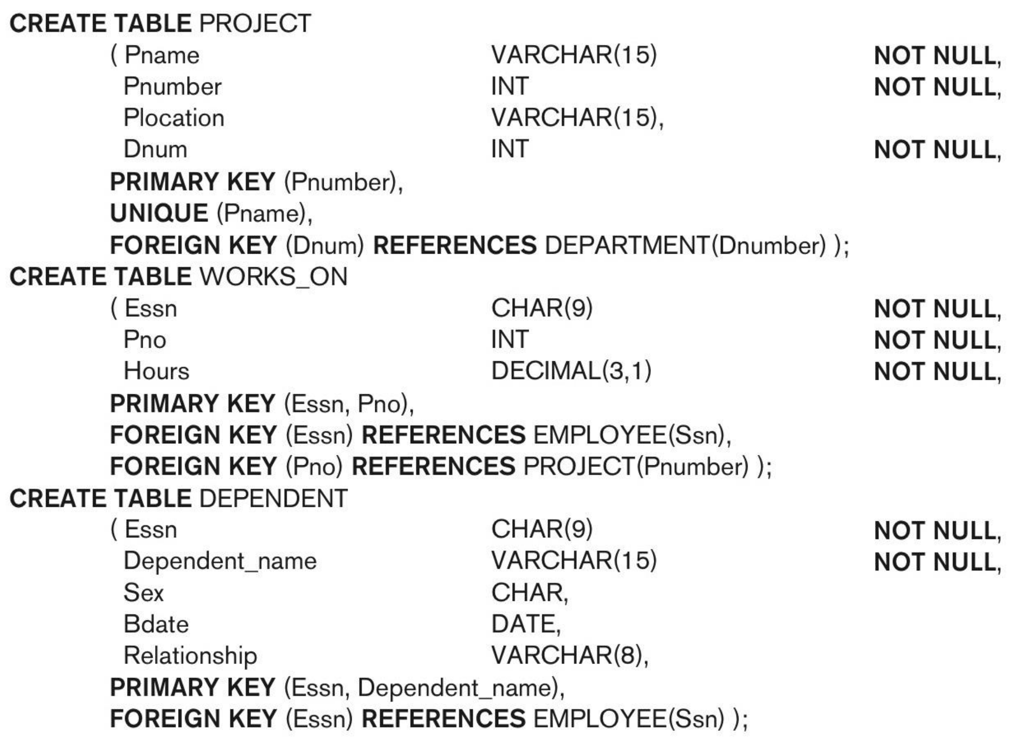
```
create table EMPLOYEES (
  empno                  number,
  name                   varchar2(50) not null,
  job                    varchar2(50),
  manager                number,
  hiredate               date,
  salary                 number(7,2),
  commission             number(7,2),
  deptno                number,
  constraint pk_employees primary key (empno),
  constraint fk_employees_deptno foreign key (deptno)
      references DEPARTMENTS (deptno)
);
```

Foreign keys must reference primary keys, so to create a "child" table the "parent" table must have a primary key for the foreign key to reference.

1. Creating Relations using SQL: *Additional examples*

```
CREATE TABLE EMPLOYEE
          ( Fname                VARCHAR(15)          NOT NULL,
            Minit                CHAR,
            Lname                VARCHAR(15)          NOT NULL,
            Ssn                  CHAR(9)              NOT NULL,
            Bdate                DATE,
            Address              VARCHAR(30),
            Sex                  CHAR,
            Salary               DECIMAL(10,2),
            Super_ssn            CHAR(9),
            Dno                  INT                  NOT NULL,
          PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
          ( Dname                VARCHAR(15)          NOT NULL,
            Dnumber              INT                  NOT NULL,
            Mgr_ssn              CHAR(9)              NOT NULL,
            Mgr_start_date       DATE,
          PRIMARY KEY (Dnumber),
          UNIQUE (Dname),
          FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
          ( Dnumber              INT                  NOT NULL,
            Dlocation            VARCHAR(15)          NOT NULL,
          PRIMARY KEY (Dnumber, Dlocation),
          FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

1. Creating Relations using SQL: *Additional examples*



```
CREATE TABLE PROJECT
        ( Pname                 VARCHAR(15)             NOT NULL,
          Pnumber               INT                     NOT NULL,
          Plocation             VARCHAR(15),
          Dnum                  INT                     NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn                  CHAR(9)                 NOT NULL,
          Pno                   INT                     NOT NULL,
          Hours                 DECIMAL(3,1)            NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn                  CHAR(9)                 NOT NULL,
          Dependent_name        VARCHAR(15)             NOT NULL,
          Sex                   CHAR,
          Bdate                 DATE,
          Relationship          VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

2. Querying Relation

- **How to query a table**

- **Statement**

SELECT <*attributes*>
FROM <*one or more relations*>
WHERE <*conditions*>

2. Querying Relation

- **A. *star (*) to select all the attributes***

Item_Name	Price	Category
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

```
SELECT *
FROM Items
WHERE category="Food"
```

Item_Name	Price	Category
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

2. Querying Relation

B. *Specifying attributes to select*

Item_Name	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

$\rightarrow$

```
SELECT Item_Name, Price
FROM Items
WHERE category="Food"
```

Item_Name	**Price**
Plain Donut	3.50
Chocolate Donut	4.50

2. Querying Relation

C. More complex *where* clause

Item_Name	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

SELECT Item_Name, Price
FROM Items
WHERE category="Food" **AND** price < 4.00

Item_Name	**Price**
Plain Donut	3.50

2. Querying Relation

D. *String Pattern Matching*

Item_Name	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

SELECT Item_Name, Price
FROM Items
WHERE Item_Name **LIKE** "%Coffee"

Item_Name	**Price**
Small Coffee	8.00
Ice Coffee	12.00

- "**%Coffee**" will return all strings that end with Coffee regardless to what comes before that.
- **"%D%**" will return all the items that has the letter "D"

2. Querying Relation

E. *Sorting the results*

Item_Name	Price	Category
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

SELECT Item_Name, Price
FROM Items
ORDER BY Price

Item_Name	Price
Plain Donut	3.50
Chocolate Donut	4.50
Small Coffee	8.00
Ice Coffee	12.00

2. Querying Relation

F. Selecting only unique values

Item_Name	Price	Category
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

```
SELECT DSITINCT Category
FROM Items
```

$\rightarrow$

Category
Drinks
Food

3. Basic Joins

- **Merging multiple tables, and querying them as if they where one**

- **Statement**

SELECT *<attributes>*
FROM *R1* JOIN *R2*
ON *R1.attribute* = *R2.attribute*
WHERE *<conditions>*

- **There are many different types of joins, and various ways of connecting tables.**

3. Basic Joins

Students

Student_ID	Student_Name	Date_of_Birth
438144933	Ahmad Fahad	1/4/1990
437555121	Sarah Khlaid	5/1/1991
436122555	Saad Nassir	9/9/1991

Students_Grades

Section_ID	S_ID	Grade
7883	438144933	A
7883	437555121	A+
3441	438144933	C

Q) What if you want to show the names, student IDs, and grade for students who were in the 7883 section?

3. Basic Joins

Query:

SELECT S_ID, Student_Name, Grade
FROM Students **JOIN** Students_Grades
ON Students.Student_ID = Students_Grades.S_ID
WHERE section_ID=‘7883’

S_ID	**Student_Name**	**Grade**
438144933	Ahmad Fahad	A
437555121	Sarah Khlaid	A+

Part 2: Insert, Update, Delete and aggregate functions

4. Insert Operations

- **Insert is used to add rows (tuples or records) to a relation (table).**

- Values are ordered in the same way the attributes are ordered
- Or you can specify the order for the attributes

- You can insert multiple rows

- **General Statement**

INSERT INTO <*relation*>
VALUES <*values for the attributes in the relation*>
WHERE <*conditions*>

4. Insert Operations

- ***Example:***

ITEMS

<u>**Item_Name**</u>	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

INSERT INTO Items
VALUES (“Small Water”,”1.00”,”Drinks”)

4. Insert Operations

- ***Example:***

SELECT *
FROM items

ITEMS

<u>**Item_Name**</u>	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food
Small Water	1.00	Drinks

4. Insert Operations

- Constraints are enforced (insert operations may fail)

4. Insert Operations

- Constraints are enforced (insert operations may fail)
- ***Example:***

ITEMS

<u>**Item_Name**</u>	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

INSERT INTO Items
VALUES ("Small Coffee",9.00,"Drinks")

Fails because the PK has to be unique

4. Insert Operations

STUDENTS

Student_ID (PK)	**Student_Name**	**Date_of_ Birth**
438144933	Ahmad Fahad	1/4/1990
437555121	Sarah Khlaid	5/1/1991
436122555	Saad Nassir	9/9/1991

STUDENTS_GRADES

Section_ID	**S_ID (FK)**	**Grade**
7883	438144933	A
7883	437555121	A+
3441	438144933	C

INSERT INTO Students_Grades
VALUES (“3441”,”436555141”,”B+”)

Fails because S_ID is a FK referencing Student_ID and the ID “436555141” does not exist in the STUDENTS table

5. Update Operations

- **Used to change values that exist in a table**

- **Statement**

UPDATE *<relation>*
SET *<attribute=new value>*
WHERE *<conditions>*

5. Update Operations

- ***Example 1: Updating one row***

Item_Name	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food

```
UPDATE Items
SET Price=9.00
WHERE Item_Name='Small Coffee'
```

Item_Name	**Price**	**Category**
Small Coffee	9.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food

5. Update Operations

- ***Example 2: Make sure you use the 'where' clause***

Item_Name	Price	Category
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food

UPDATE Items
SET Price=9

Item_Name	Price	Category
Small Coffee	9.00	Drinks
Ice Coffee	9.00	Drinks
Plain Donut	9.00	Food

All rows get updated

5. Update Operations

- ***Example 3: Updating multiple rows***

Item_Name	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food

```
UPDATE Items
SET Price=Price + 1
WHERE Category='Drinks'
```

Item_Name	**Price**	**Category**
Small Coffee	9.00	Drinks
Ice Coffee	13.00	Drinks
Plain Donut	3.50	Food

5. Update Operations

- ***Example 4: Updating multiple attributes***

Item_Name	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food

UPDATE Items
SET Price=Price + 1, category="Hot Drinks"
WHERE Category='Drinks'

Item_Name	**Price**	**Category**
Small Coffee	9.00	Hot Drinks
Ice Coffee	**13.00**	**Hot Drinks**
Plain Donut	3.50	Food

6. Delete Operations

- **How to delete rows from tables**

- **Statement**

DELETE FROM *<relation>*
WHERE *<conditions>*

6. Delete Operations

- ***Example***

Item_Name	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food

DELETE FROM Items
WHERE Category=‘Food’

after a select operation

Item_Name	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks

7. Aggregate Functions

- Used to summarize information from multiple tuples into a single-tuple summary

- **Built-in aggregate functions:**

1. **Count:** Number of rows based on a condition
2. **AVG:** Average value for an attribute
3. **Min:** Minimum value for an attribute
4. **Max:** Maximum value for an attribute
5. **Sum:** Total sum of an attribute

- ***You can use 'group by' with these functions to group the results using one or more attributes***

7. Aggregate Functions

A. Count

Item_Name	Price	Category
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

SELECT COUNT (Item_Name)
FROM Items

Results= 4

SELECT COUNT (Item_Name)
FROM Items
WHERE Category='Drinks'

Results= 2

7. Aggregate Functions

B. *Min and Max*

Item_Name	Price	Category
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

SELECT MIN (Price)
FROM Items

Results= 3.50

SELECT MAX (Price)
FROM Items
WHERE Category='Food'

Results= 4.5

7. Aggregate Functions

C. *Sum*

Item_Name	**Price**	**Category**
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

```
SELECT SUM (Price)
FROM Items
```

Results= 28

```
SELECT SUM (Price)
FROM Items
WHERE Category=‘Food’
```

Results= 8

7. Aggregate Functions

D. *Group by*

Item_Name	Price	Category
Small Coffee	8.00	Drinks
Ice Coffee	12.00	Drinks
Plain Donut	3.50	Food
Chocolate Donut	4.50	Food

```
SELECT Category, SUM (Price)
FROM Items
GROUP BY Category
```

Category	Price
Drinks	20.00
Food	8.00

Summary

1. Create Table
2. Querying Relations
3. Basic Joins
4. Insert
5. Update
6. Delete
7. Aggregate Functions