IT360: Applied Database Systems

SQL: Structured Query Language
(Chapter 2 and 7 in Kroenke book)

Goals
- SQL: Data Definition Language
  - CREATE
  - ALTER
  - DROP
- SQL: Data Manipulation Language
  - INSERT
  - DELETE
  - UPDATE
  - SELECT

Relational Query Languages
- A major strength of the relational model:
  - supports simple, powerful querying of data
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.

SQL DDL and DML
- SQL statements can be divided into two categories:
  - Data definition language (DDL) statements
    - Used for creating and modifying tables, views, and other structures
    - CREATE, DROP, ALTER
  - Data manipulation language (DML) statements.
    - Used for queries and data modification
    - INSERT, DELETE, UPDATE, SELECT
Creating Tables

CREATE TABLE table_name(
    column_name1 column_type1 [constraints1],
    ...
) [CONSTRAINT constraint_name] table_constraint

Table constraints:
- NULL/NOT NULL
- PRIMARY KEY (columns)
- UNIQUE (columns)
- CHECK (conditions)
- FOREIGN KEY (local_columns) REFERENCES foreign_table (foreign_columns) [ON DELETE action_d ON UPDATE action_u]

Specify surrogate key in SQL Server:
    column_name int_type IDENTITY (seed, increment)
Specify surrogate key in MySQL:
    column_name int_type AUTO_INCREMENT

FOREIGN KEY Constraints

CREATE TABLE Departments
    (DepartmentName char(18),
     Phone char(18) NOT NULL,
     Building char(18),
     Room integer,
     PRIMARY KEY (DepartmentName)
)

CREATE TABLE Students
    (StudentNumber integer NOT NULL,
     StudentLastName char(18) NOT NULL,
     StudentFirstName char(18) NOT NULL,
     Email varchar(50) NOT NULL,
     PhoneNumber char(18) NOT NULL,
     MajorDepartmentName char(18),
     PRIMARY KEY (StudentNumber),
     CONSTRAINT U_Email UNIQUE (Email),
     CONSTRAINT FK_Dept FOREIGN KEY(MajorDepartmentName) REFERENCES DEPARTMENTS(DepartmentName) ON DELETE NO ACTION ON UPDATE CASCADE
)

FOREIGN KEY Constraints

- 4 options on deletes and updates:
  - NO ACTION (delete/update is rejected)
  - CASCADE
  - SET NULL
  - SET DEFAULT

CREATE TABLE Departments
    (DepartmentName char(18),
     Phone char(18) NOT NULL,
     Building char(18),
     Room integer,
     PRIMARY KEY (DepartmentName)
)

CREATE TABLE Students
    (StudentNumber integer,
     StudentLastName char(18) NOT NULL,
     StudentFirstName char(18) NOT NULL,
     Email char(50) NOT NULL,
     PhoneNumber char(18) NOT NULL,
     MajorDepartmentName char(18),
     CONSTRAINT PK_Students PRIMARY KEY (StudentNumber),
     CONSTRAINT U_Email UNIQUE (Email),
     CONSTRAINT FK_Dept FOREIGN KEY(MajorDepartmentName) REFERENCES DEPARTMENTS(DepartmentName) ON DELETE SET NULL ON UPDATE CASCADE
)
Modifying Tables

- **ALTER TABLE** table_name clause

   **Clauses:**
   - **ADD COLUMN** column_name column_type [constraints]
   - **DROP COLUMN** column_name
   - **ALTER COLUMN / MODIFY** – DBMS specific!
   - **ADD CONSTRAINT** constraint
   - **DROP CONSTRAINT** constraint_name

ALTER TABLE Examples

- **ALTER TABLE Students** ADD COLUMN BirthDate
datetime NULL
- **ALTER TABLE Students** DROP COLUMN BirthDate
- **ALTER TABLE Student** ADD CONSTRAINT FK_Department
  FOREIGN KEY (MajorDepartmentName)
  REFERENCES Departments (DepartmentName)
  ON DELETE NO ACTION
  ON UPDATE CASCADE

Removing Tables

- **DROP TABLE** table_name

  **DROP TABLE** Departments;

  If there are constraints dependent on table:
  - Remove constraints
  - Drop table

  **ALTER TABLE Students**
  **DROP CONSTRAINT** FK_Department;

  **DROP TABLE** Departments;

SQL DDL and DML

- **Data definition language (DDL)** statements
  - Used for creating and modifying tables, views, and other structures
  - **CREATE, ALTER, DROP**

- **Data manipulation language (DML)** statements.
  - Used for queries and data modification
  - **INSERT, DELETE, UPDATE, SELECT**
SQL DML

- Data manipulation language (DML) statements.
  - Used for queries and data modification
  - INSERT
  - DELETE
  - UPDATE
  - SELECT

### INSERT Statement

**INSERT INTO** table_name [(column_list)] VALUES (data_values)

**INSERT INTO** table_name [column_list] select_statement

**INSERT command:**

```
INSERT INTO Students (StudentNumber, StudentLastName, StudentFirstName)
VALUES (190, 'Smith', 'John');
```

**Bulk INSERT:**

```
INSERT INTO Students (StudentNumber, StudentLastName, StudentFirstName, Email, PhoneNumber)
SELECT *
FROM Second_Class_Students;
```

### UPDATE Statement

UPDATE table_name
SET column_name1 = expression1 [column_name2 = expression2,...] [WHERE search_condition]

**UPDATE command:**

```
UPDATE Students
SET PhoneNumber = '410-123-4567'
WHERE StudentNumber = 673;
```

**BULK UPDATE command:**

```
UPDATE Students
SET PhoneNumber = '410-123-4567'
WHERE StudentLastName = 'Doe';
```

### DELETE Statement

DELETE FROM table_name
[WHERE search_condition]

**DELETE command:**

```
DELETE FROM Students
WHERE StudentNumber = 190;
```

If you omit the WHERE clause, you will delete every row in the table!!

**Another example:**

```
DELETE FROM Departments
WHERE DepartmentName = 'ComSci'
```

Integrity constraints?!
The SQL SELECT Statement

- Basic SQL Query:
  ```sql
  SELECT [DISTINCT] column_name(s) | *
  FROM table_name(s)
  [WHERE conditions]
  ```

Selecting All Columns: The Asterisk (*) Keyword

```sql
SELECT *
FROM Students;
```

<table>
<thead>
<tr>
<th>StudentNumber</th>
<th>StudentFirstName</th>
<th>Email</th>
<th>PhoneNumber</th>
<th>MajDeptName</th>
</tr>
</thead>
<tbody>
<tr>
<td>190</td>
<td>Smith</td>
<td><a href="mailto:smith@usa.edu">smith@usa.edu</a></td>
<td>410-431-3405</td>
<td>ComSci</td>
</tr>
<tr>
<td>673</td>
<td>Doe</td>
<td><a href="mailto:doe@usa.edu">doe@usa.edu</a></td>
<td></td>
<td>ComSci</td>
</tr>
<tr>
<td>312</td>
<td>Doe</td>
<td><a href="mailto:doe@usa.edu">doe@usa.edu</a></td>
<td>443-451-7665</td>
<td>Math</td>
</tr>
</tbody>
</table>

Specific Columns and Rows from One Table

```sql
SELECT StudentNumber, StudentLastName, StudentFirstName
FROM Students
WHERE MajDeptName = 'ComSci';
```

<table>
<thead>
<tr>
<th>StudentNumber</th>
<th>StudentLastName</th>
<th>StudentFirstName</th>
</tr>
</thead>
<tbody>
<tr>
<td>190</td>
<td>Smith</td>
<td>John</td>
</tr>
<tr>
<td>673</td>
<td>Doe</td>
<td>Jane</td>
</tr>
</tbody>
</table>

The DISTINCT Keyword

```sql
SELECT SName
FROM Students;
```

<table>
<thead>
<tr>
<th>StudentLastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doe</td>
</tr>
<tr>
<td>Smith</td>
</tr>
</tbody>
</table>

```sql
SELECT DISTINCT SName
FROM Students;
```

<table>
<thead>
<tr>
<th>StudentLastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doe</td>
</tr>
<tr>
<td>Smith</td>
</tr>
</tbody>
</table>
Class Exercise

- Division(Name, Building, OfficeNb)
- Department(DeptName, ChairName, WebAddress, DivName)

- Create tables
- Modify Department to add a FK constraint for DivName
- Create table Colleges with same structure as Division
- Insert everything from Division into Colleges
- Remove Division table
- Find the name of the Chair of the ‘Math’ Department

SELECT from Two or More Tables

Find the names of students enrolled in IT360

```sql
SELECT SName
FROM Students S, Enrolled E
WHERE S.Snb = E.SNb AND E.Cid = 'IT360'
```

Example Conceptual Evaluation

```sql
SELECT SName
FROM Students S, Enrolled E
WHERE S.Snb = E.SNb AND E.Cid = 'IT360'
```

SELECT - Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
- Compute the cross-product of table_names
- Discard resulting rows if they fail condition
- Delete columns that are not in column_names
- If DISTINCT is specified, eliminate duplicate rows
- This strategy is probably the least efficient way to compute a query!
- An optimizer will find more efficient strategies to compute the same answers.
Example Conceptual Evaluation

SELECT SName
FROM Students S, Enrolled E
WHERE S.Snb = E.SNb AND E.Cid = 'IT360'

<table>
<thead>
<tr>
<th>S.Nb</th>
<th>SName</th>
<th>Email</th>
<th>E.SNb</th>
<th>Cid</th>
<th>Semester</th>
</tr>
</thead>
<tbody>
<tr>
<td>190</td>
<td>Smith</td>
<td><a href="mailto:jdoe@usna.edu">jdoe@usna.edu</a></td>
<td>190</td>
<td>IT340</td>
<td>Spring2005</td>
</tr>
<tr>
<td>190</td>
<td>Smith</td>
<td><a href="mailto:jdoe@usna.edu">jdoe@usna.edu</a></td>
<td>190</td>
<td>IT340</td>
<td>Fall2005</td>
</tr>
<tr>
<td>673</td>
<td>Doe</td>
<td><a href="mailto:jdoe@usna.edu">jdoe@usna.edu</a></td>
<td>312</td>
<td>IT360</td>
<td>Spring2005</td>
</tr>
<tr>
<td>312</td>
<td>Doe</td>
<td><a href="mailto:jdoe@usna.edu">jdoe@usna.edu</a></td>
<td>312</td>
<td>IT360</td>
<td>Fall2005</td>
</tr>
<tr>
<td>312</td>
<td>Doe</td>
<td><a href="mailto:jdoe@usna.edu">jdoe@usna.edu</a></td>
<td>312</td>
<td>IT360</td>
<td>Fall2005</td>
</tr>
</tbody>
</table>

Modified Query

SELECT SNb
FROM Students S, Enrolled E
WHERE S.Snb = E.SNb AND E.Cid = 'IT360'

- Would the result be different with DISTINCT?

Example Conceptual Evaluation

SELECT SName
FROM Students S, Enrolled E
WHERE S.Snb = E.SNb AND E.Cid = 'IT360'

<table>
<thead>
<tr>
<th>S.Nb</th>
<th>SName</th>
<th>Email</th>
<th>E.SNb</th>
<th>Cid</th>
<th>Semester</th>
</tr>
</thead>
<tbody>
<tr>
<td>190</td>
<td>Smith</td>
<td><a href="mailto:jdoe@usna.edu">jdoe@usna.edu</a></td>
<td>190</td>
<td>IT340</td>
<td>Spring2005</td>
</tr>
<tr>
<td>190</td>
<td>Smith</td>
<td><a href="mailto:jdoe@usna.edu">jdoe@usna.edu</a></td>
<td>190</td>
<td>IT340</td>
<td>Fall2005</td>
</tr>
<tr>
<td>673</td>
<td>Doe</td>
<td><a href="mailto:jdoe@usna.edu">jdoe@usna.edu</a></td>
<td>312</td>
<td>IT360</td>
<td>Spring2005</td>
</tr>
<tr>
<td>312</td>
<td>Doe</td>
<td><a href="mailto:jdoe@usna.edu">jdoe@usna.edu</a></td>
<td>312</td>
<td>IT360</td>
<td>Fall2005</td>
</tr>
<tr>
<td>312</td>
<td>Doe</td>
<td><a href="mailto:jdoe@usna.edu">jdoe@usna.edu</a></td>
<td>312</td>
<td>IT360</td>
<td>Fall2005</td>
</tr>
</tbody>
</table>

Class Exercise

- Students(SNb, SName, Email)
- Courses(Cid, CName, Dept)
- Enrolled(SNb, Cid, Semester)

- Find the student number and name for each student enrolled in ‘Spring2009’ semester
- Find the names of all students enrolled in ‘ComSci’ courses
Sorting the Results

SELECT [DISTINCT] column_name(s) | * 
FROM table_name(s) 
[WHERE conditions] 
[ORDER BY column_name(s)[ASC/DESC]]

Example:
Students(SNb, SName, Email, Major)

SELECT SNb, SName 
FROM Students 
ORDER BY SName ASC, SNb DESC

WHERE Clause Options

- AND, OR
- IN, NOT IN, BETWEEN
- LIKE

Wild cards:
- * = Exactly one character
- % = Any set of one or more characters

MS Access
- ? = Exactly one character
- _ = Any set of one or more characters

Example:
Students(SNb, SName, Email, Major)
Find alpha and name of SCS or SIT students with SNb starting with '9'

SELECT SNb, SName 
FROM Students 
WHERE SNb LIKE '9%' AND 
Major IN ('SIT', 'SCS')

Class Exercise

- Students(SNb, SName, Email)
- Courses(Cid, CName, Dept)
- Enrolled(SNb, Cid, Semester)

Find the student number and name for each student enrolled in 'Spring2008' semester
Find the names of all students enrolled in 'ComSci' courses

Calculations in SQL

- Simple arithmetic
- Five SQL Built-in Functions:
  - COUNT
  - SUM
  - AVG
  - MIN
  - MAX
Simple Arithmetic

- SELECT NbHours * HourlyRate AS Charge FROM FlightEvents
- SELECT SFirstName + ' ' + SLastName FROM Students

<table>
<thead>
<tr>
<th>Charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
</tr>
<tr>
<td>400</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>400</td>
</tr>
</tbody>
</table>

Aggregate Operators

- SELECT COUNT(*) FROM Students
- SELECT COUNT(DISTINCT SName) FROM Students WHERE SNb > 700
- SELECT AVG(Age) FROM Students WHERE SNb LIKE '09____'

Aggregate Operators Limitations

- Return only one row
- Not in WHERE clause

Select oldest students and their age

- SELECT S.SName, MAX (Age) FROM Students S
- SELECT S.SName, S.Age FROM Students S WHERE S.AGE = (SELECT MAX(Age) FROM Students)
Select students with age higher than average

- SELECT *
  FROM Students
  WHERE Age > AVG(Age)

  Illegal!

- SELECT *
  FROM Students
  WHERE Age > (SELECT AVG(Age)
              FROM Students)

  Correct!

Class Exercise

- Students(SNb, SName, Email)
- Courses(Cid, CName, Dept)
- Enrolled(SNb, Cid, Semester)

List SNb of all students enrolled in ‘IT360’ or ‘IT340’, ordered by SNb

Grouping rows

- Find the age of the youngest student for each class year
- SELECT MIN (S.Age)
  FROM Students S
  WHERE S.ClassYear = 2009

  (no column name)
  21

GROUP-BY Clause

- SELECT [DISTINCT] column_name(s) | aggregate_expr
  FROM table_name(s)
  [WHERE conditions]
  GROUP BY grouping_columns

- Example:
  SELECT ClassYear, MIN(Age)
  FROM Students
  GROUP BY ClassYear

<table>
<thead>
<tr>
<th>ClassYear</th>
<th>(no column name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>21</td>
</tr>
<tr>
<td>2012</td>
<td>17</td>
</tr>
<tr>
<td>2011</td>
<td>18</td>
</tr>
<tr>
<td>2010</td>
<td>20</td>
</tr>
</tbody>
</table>
Conceptual Evaluation

- Semantics of an SQL query defined as follows:
  - Compute the cross-product of tables in FROM (table_names)
  - Discard resulting rows if they fail WHERE conditions
  - Delete columns that are not in SELECT or GROUP BY(column_names or grouping-columns)
  - Remaining rows are partitioned into groups by the value of the columns in grouping-columns
  - One answer row is generated per group
- Note: Does not imply query will actually be evaluated this way!

HAVING Clause

- SELECT [DISTINCT] column_name(s) | aggregate_expr
  FROM table_name(s)
  [WHERE conditions]
  GROUP BY grouping_columns
  HAVING group_conditions
  - GROUP BY groups the rows
  - HAVING restricts the groups presented in the result

Example- HAVING

- SELECT ClassYear, MIN(Age)
  FROM Students
  WHERE MajDeptName = 'ComSci'
  GROUP BY ClassYear
  HAVING COUNT(*) > 20

Conceptual Evaluation

- SQL query semantics:
  - Compute the cross-product of table_names
  - Discard resulting rows if they fail conditions
  - Delete columns that are not specified in SELECT, GROUP BY
  - Remaining rows are partitioned into groups by the value of the columns in grouping-columns
  - One answer row is generated per group
  - Discard resulting groups that do not satisfy group_conditions
Example

- SELECT Class, MIN(Age)
  FROM Students
  WHERE MajDeptName = 'ComSci'
  GROUP BY Class
  HAVING COUNT(*) > 2

Class Exercise

- Students(SNb, SName, Email)
- Courses(Cid, CName, Dept)
- Enrolled(SNb, Cid, Semester)

  List all course names, and the number of students enrolled in the course

Subqueries

- SELECT *
  FROM Students
  WHERE Age > (SELECT AVG(Age)
    FROM Students)

- Second select is a subquery (or nested query)

- You can have subqueries in FROM or HAVING clause also

Subqueries in FROM Clause

- Find name of students enrolled in both 'IT360' and 'IT334'

- SELECT FName + ' ' + LName AS StudentName
  FROM Students, (SELECT Alpha
    FROM Enroll
    WHERE CourseID = 'IT360'
    AND Alpha IN
    (SELECT Alpha
      FROM Enroll
      WHERE CourseID = 'IT334')
  ) AS ResultAlphaTable
  WHERE Students.Alpha = ResultAlphaTable.Alpha
Subqueries Exercise

- Students(Alpha, LName, FName, Class, Age)
- Enroll(Alpha, CourseID, Semester, Grade)
1. Find alpha for students enrolled in both ‘IT360’ and ‘IT334’
2. Find name of students enrolled in both ‘IT360’ and ‘IT334’

Class Exercise

- Students(Alpha, LName, FName, Class, Age)
- Enroll(Alpha, CourseID, Semester, Grade)

- Find the name of students enrolled in ‘IT360’
  - Usual way
  - Use subqueries

Class Exercise

- What does this query compute:
- SELECT FName, LName
  FROM Students S, Enroll E1, Enroll E2
  WHERE S.Alpha = E1.Alpha
    AND S.Alpha = E2.Alpha
    AND E1.CourseID = ‘IT360’
    AND E2.CourseID = ‘IT344’

Summary

- SELECT [DISTINCT] column_name(s) | aggregate_expr
  FROM table_name(s)
  WHERE conditions
  GROUP BY grouping_columns
  HAVING group_conditions
  ORDER BY column_name(s) [ASC/DESC]