



Introductory SQL, Part 1

Create Databases and Select Data

Part 1 of Chapter 3 in the Text Book

©Anne Haxthausen and Flemming Schmidt

These slides have been prepared by Anne Haxthausen, using a figure from Thorbjørn Konstantinovitz and partly reusing/modifying slides by Flemming Schmidt, which again partly reused some slides by Silberschatz, Korth. Sudarshan, 2010.

Contents

- University Database (running example)
- SQL Command Categories (book sec. 3.1)
- SQL Data Definition (book sec. 3.2)
- SQL Data Manipulation (book sec. 3.9, except parts using subqueries)
- SQL Data Queries (book sec. 3.3-3.4, except 3.4.2) + (sec. 4.1.1 in 7th edition = 3.3.4 in 6th)
- Making New Tables From Old Tables
- Demo Exercises & Exercises



University Database

- Database example used throughout the textbook and in this course!

1. Statement of Requirements

- Contains details about entities and their relationship and related tasks needed to run a university. Often established via client interviews.

2. Database Schema

- Contains information of the database in a listing of Relation Schemas with attribute names and primary keys.

3. Database Schema Diagram

- Contains information of relations, attributes, primary keys and foreign keys in a diagram format.

4. SQL Data Definition

5. Database Instance

- Shows the relation contents at a specific moment in time

1 - Statement of Requirements

(see also Section 1.6.2 in the textbook)

- Based on client interviews

The university is organized in named **departments**.

Departments have assigned yearly **budgets** and are located in **buildings**.

Each department employs named **instructors**.

Instructors have assigned **salaries**.

Each department has named **students**.

Each student has a **birthday**, total **credits** for passed examinations, and can have one department **advisor** assigned.

Each department offers named **courses**.

Each course is offered each **year** in defined **semesters**. Each offering of a course is called a **section** and is **taught** by instructors and **taken** by students, who gets a credit for passing the examination. Courses may have other courses as **prereq**. Sections are conducted in a weekly **timeslot**, in a specific campus **class room** (**building room**), which has a maximum student **capacity**.

And more about who uses the database and for what purpose...

2 - Database Schema

■ A set of Relation Schemas

Each with a Relation Name, Attribute Names and Primary Key Attributes underlined

Classroom(Building, Room, Capacity)

Department(DeptName, Building, Budget)

Course(CourseID, Title, DeptName, Credits)

Instructor(InstID, InstName, DeptName, Salary)

Section(CourseID, SectionID, Semester, StudyYear, Building, Room, TimeSlotID)

Teaches(InstID, CourseID, SectionID, Semester, StudyYear)

Student(StudID, StudName, Birth, DeptName, TotCredits)

Takes(StudID, CourseID, SectionID, Semester, StudyYear, Grade)

Advisor(StudID, InstID)

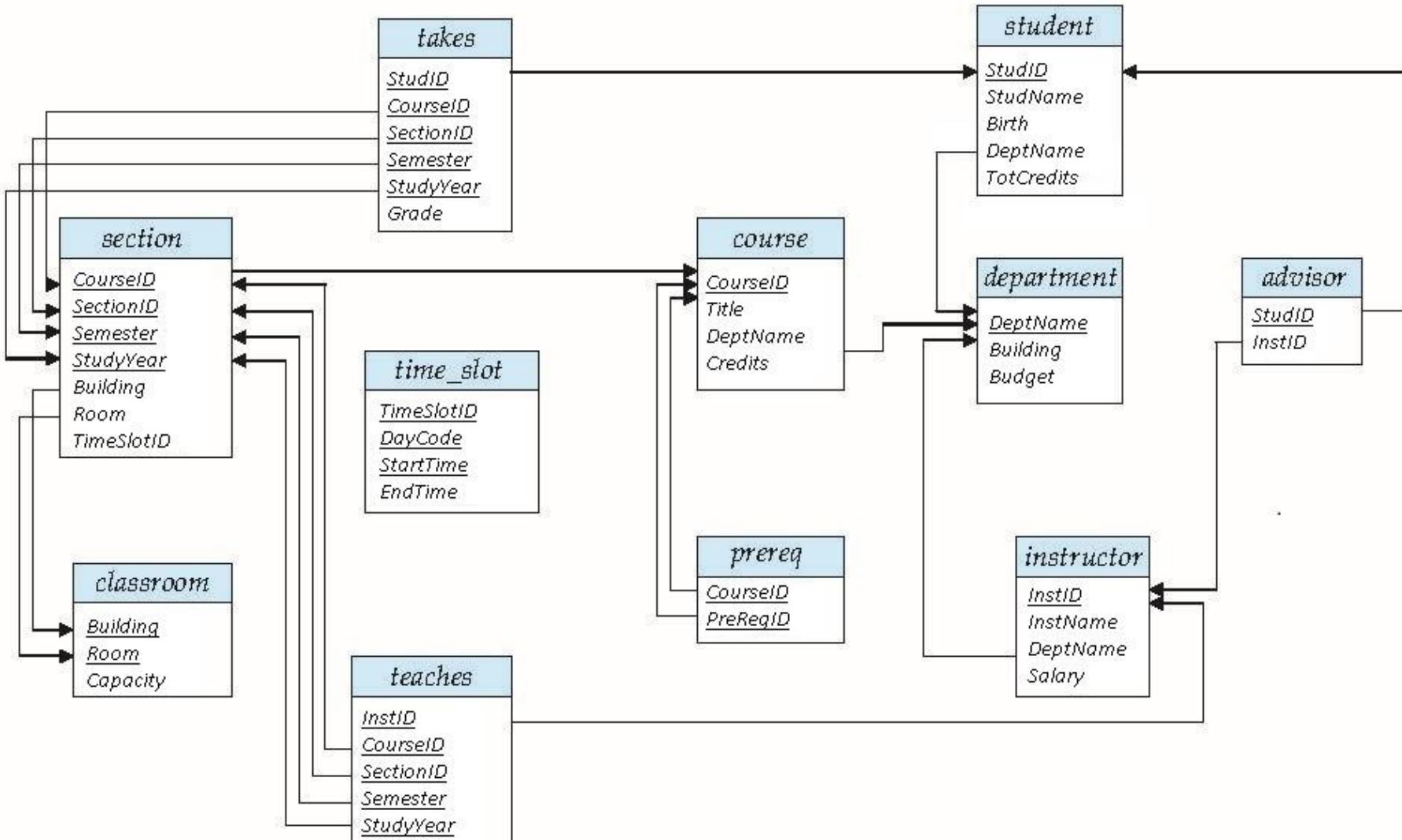
TimeSlot(TimeSlotID, DayCode, StartTime, EndTime)

PreReq(CourseID, PreReqID)

■ Challenge

- How to give tables and attributes short but meaningful names
- E.g. How to name the ID attribute of an instructor: InstructorID, IID, Inst_ID or InstID?
- Here some of the names differ slightly from those in the textbook and Student has one extra attribute: Birth (for birthday).

3 - Database Schema Diagram from the book, but with new names



4 - SQL Data Definition

- SQL CREATE TABLE statements for each of the tables can be found in the **UniversityDB.sql** script on DTU Learn under *Content → Databases* .

5 - Database Instance (1 of 4)

Instructor

InstID	InstName	DeptName	Salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

Student

StudID	StudName	Birth	DeptName	TotCredits
00128	Zhang	1992-04-18	Comp. Sci.	102
12345	Shankar	1995-12-06	Comp. Sci.	32
19991	Brandt	1993-05-24	History	80
23121	Chavez	1992-04-18	Finance	110
44553	Peltier	1995-10-18	Physics	56
45678	Levy	1995-08-01	Physics	46
54321	Williams	1995-02-28	Comp. Sci.	54
55739	Sanchez	1995-06-04	Music	38
70557	Snow	1995-11-22	Physics	0
76543	Brown	1994-03-05	Comp. Sci.	58
76653	Aoi	1993-09-18	Elec. Eng.	60
98765	Bourikas	1992-09-23	Elec. Eng.	98
98988	Tanaka	1992-06-02	Biology	120

Advisor

StudID	InstID
12345	10101
44553	22222
45678	22222
00128	45565
76543	45565
23121	76543
98988	76766
76653	98345
98765	98345

Department

DeptName	Building	Budget
Biology	Watson	90000.00
Comp. Sci.	Taylor	100000.00
Elec. Eng.	Taylor	85000.00
Finance	Painter	120000.00
History	Painter	50000.00
Music	Packard	80000.00
Physics	Watson	70000.00

Database Instance (2 of 4)

Teaches

InstID	CourseID	SectionID	Semester	StudyYear
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
10101	CS-101	1	Fall	2009
45565	CS-101	1	Spring	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
10101	CS-315	1	Spring	2010
45565	CS-319	1	Spring	2010
83821	CS-319	2	Spring	2010
10101	CS-347	1	Fall	2009
98345	EE-181	1	Spring	2009
12121	FIN-201	1	Spring	2010
32343	HIS-351	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

Takes

StudID	CourseID	SectionID	Semester	StudyYear	Grade
00128	CS-101	1	Fall	2009	A
00128	CS-347	1	Fall	2009	A-
12345	CS-101	1	Fall	2009	C
12345	CS-190	2	Spring	2009	A
12345	CS-315	1	Spring	2010	A
12345	CS-347	1	Fall	2009	A
19991	HIS-351	1	Spring	2010	B
23121	FIN-201	1	Spring	2010	C+
44553	PHY-101	1	Fall	2009	B-
45678	CS-101	1	Fall	2009	F
45678	CS-101	1	Spring	2010	B+
45678	CS-319	1	Spring	2010	B
54321	CS-101	1	Fall	2009	A-
54321	CS-190	2	Spring	2009	B+
55739	MU-199	1	Spring	2010	A-
76543	CS-101	1	Fall	2009	A
76543	CS-319	2	Spring	2010	A
76653	EE-181	1	Spring	2009	C
98765	CS-101	1	Fall	2009	C-
98765	CS-315	1	Spring	2010	B
98988	BIO-101	1	Summer	2009	A
98988	BIO-301	1	Summer	2010	NULL

Database Instance (3 of 4)

Course

CourseID	Title	DeptName	Credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

PreReq

CourseID	PreReqID
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Database Instance (4 of 4)

Section

CourseID	SectionID	Semester	StudyYear	Building	Room	TimeSlotID
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Classroom

Building	Room	Capacity
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

TimeSlot

TimeSlotID	DayCode	StartTime	EndTime
A	M	08:00:00	08:50:00
A	W	08:00:00	08:50:00
A	F	08:00:00	08:50:00
B	M	09:00:00	09:50:00
B	W	09:00:00	09:50:00
B	F	09:00:00	09:50:00
C	M	11:00:00	11:50:00
C	W	11:00:00	11:50:00
C	F	11:00:00	11:50:00
D	M	13:00:00	13:50:00
D	W	13:00:00	13:50:00
D	F	13:00:00	13:50:00
E	T	10:30:00	11:45:00
E	R	10:30:00	11:45:00
F	T	14:30:00	15:45:00
F	R	14:30:00	15:45:00
G	M	16:00:00	16:50:00
G	W	16:00:00	16:50:00
G	F	16:00:00	16:50:00
H	W	10:00:00	12:30:00

Note: In writing and testing SQL queries a print of the Diagram and the Database Instance is most useful.

SQL Command Categories (book sec 3.1)

■ SQL Command Categories with Examples

- Data Definition Language (DDL)

CREATE DATABASE, DROP DATABASE,
CREATE TABLE, ALTER TABLE, DROP TABLE,
CREATE INDEX, ALTER INDEX, DROP INDEX,
CREATE VIEW and DROP VIEW.

- Data Manipulation Language (DML)

INSERT, UPDATE and DELETE

- Data Query Language

SELECT

- Data Control Language

CREATE USER, ALTER PASSWORD, GRANT, REVOKE and CREATE SYNONYM

- Data Administration Commands

START AUDIT and STOP AUDIT

- Transactional Control Commands

SET TRANSACTION, COMMIT, ROLLBACK and SAVEPOINT



SQL Data Definition Language (DDL) (book sec 3.2)

- The DDL provides commands for changing database schemas.
- In this section you will learn about
 - The commands CREATE DATABASE, USE and DROP DATABASE
 - The commands CREATE TABLE, DROP TABLE and ALTER TABLE
 - Types/Domains in SQL
 - Constraints in SQL

CREATE DATABASE, USE and DROP DATABASE

■ Create a Database

- Create a database with the name 'University':

```
CREATE DATABASE University;
```

■ Use a Database

- Instruct the DBMS to use the University as the default (current) database for subsequent statements. The database remains the default until the end of the session or another USE statement is issued.

```
USE University;
```

■ Drop an existing Database

```
DROP DATABASE University;
```

Domain Types in SQL

- **String types**

CHAR(n)	Fixed length character strings, with user-specified length n
VARCHAR(n)	Variable length character strings, with user-specified maximum length n.

- **Date and Time**

DATE	A Date in the format 'YYYY-MM-DD'
...	

- **Numeric types**

INT	Integers, a finite subset of the integers that is machine-dependent.
SMALLINT	Small integers (a machine-dependent subset of INT).
DECIMAL(p,d)	Fixed point numbers, with user-specified precision of p digits, with d digits to the right of decimal point.
FLOAT(n)	Floating point numbers (like 1.23 and 1.23E5) with user-specified precision of at least n digits.

Many more are available – the selection depends on the DBMS implementation.

Choosing Domains wisely will enhance consistency, save storage and improve response time!

CREATE TABLE

- An SQL relation is defined using the **CREATE TABLE** command

- Typical form:
**CREATE TABLE R ($A_1 D_1, A_2 D_2, \dots, A_n D_n,$
 (integrity-constraint₁),
 ...,
 (integrity-constraint_k))**

- R is the name of the relation.
- Each A_i is an attribute name in the schema of relation R .
- D_i is the data type (domain) of values of attribute A_i .
- Possible integrity constraints include: primary key, foreign key, not null.

- Example: Creation of a Table with a Primary Key

```
CREATE TABLE Classroom  
(Building    VARCHAR(15),  
 Room        VARCHAR(7),  
 Capacity     DECIMAL(4,0),  
 PRIMARY KEY(Building, Room));
```

- A Primary Key represents an Integrity Constraint

In the example, it will ensure that all rows have

1. a unique (Building, Room) combination
2. Building and Room are not NULL.

CREATE TABLE

■ Example: creation of a table with a Foreign Key Constraint

CREATE TABLE Instructor

```
(InstID      VARCHAR(5),  
InstName    VARCHAR(20),  
DeptName    VARCHAR(20),  
Salary       DECIMAL(8,2),  
PRIMARY KEY (InstID),  
FOREIGN KEY(DeptName) REFERENCES Department(DeptName);
```

The foreign key will ensure that the DeptName
in any **Instructor** row will also appear in **Department**.

In particular this means that the DBMS should as a
default (for other possibilities see next page)
disallow

- insert violations: the insertion of a row into **Instructor**,
if its DeptName is not in the **Department** table
- delete violation: the deletion of a row from **Department**
table, if its DeptName is used in a row in **Instructor**
- update violations: update of a DeptName in **Department**,
if its DeptName is used in a row in **Instructor**

Instructor:

InstID	InstName	DeptName	Salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

Department:

DeptName	Building	Budget
Biology	Watson	90000.00
Comp. Sci.	Taylor	100000.00
Elec. Eng.	Taylor	85000.00
Finance	Painter	120000.00
History	Painter	50000.00
Music	Packard	80000.00
Physics	Watson	70000.00

CREATE TABLE

- It is possible to define referential actions for foreign key constraints, e.g.:
 - ON DELETE SET NULL
 - ON DELETE CASCADE

- Example:**

```
CREATE TABLE Instructor
```

```
(InstID      VARCHAR(5),  
 InstName    VARCHAR(20),  
 DeptName    VARCHAR(20),  
 Salary       DECIMAL(8,2),  
 PRIMARY KEY (InstID),  
 FOREIGN KEY(DeptName) REFERENCES  
 Department(DeptName) ON DELETE SET NULL);
```

Now it is allowed to delete a **Department** row having a **DeptName** used in some rows in the **Instructor** table. In that case the **DeptName** in those **Instructor** rows are set to **NULL**.

Instructor

InstID	InstName	DeptName	Salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	NULL	95000.00
32343	El Said	History	60000.00
33456	Gold	NULL	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

Department

DeptName	Building	Budget
Biology	Watson	90000.00
Comp. Sci.	Taylor	100000.00
Elec. Eng.	Taylor	85000.00
Finance	Painter	120000.00
History	Painter	50000.00
Music	Packard	80000.00
Physics	Watson	70000.00

CREATE TABLE

- It is possible to define referential actions for foreign key constraints, e.g.:
 - ON DELETE SET NULL
 - ON DELETE CASCADE

- Example:

```
CREATE TABLE Instructor
```

```
(InstID      VARCHAR(5),  
 InstName    VARCHAR(20),  
 DeptName    VARCHAR(20),  
 Salary       DECIMAL(8,2),  
 PRIMARY KEY (InstID),  
 FOREIGN KEY(DeptName) REFERENCES  
 Department(DeptName) ON DELETE CASCADE);
```

Now it is allowed to delete a **Department** row having a **DeptName** used in some rows in the **Instructor** table. In that case those **Instructor** rows are deleted too.

Instructor

InstID	InstName	DeptName	Salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

Department

DeptName	Building	Budget
Biology	Watson	90000.00
Comp. Sci.	Taylor	100000.00
Elec. Eng.	Taylor	85000.00
Finance	Painter	120000.00
History	Painter	50000.00
Music	Packard	80000.00
Physics	Watson	70000.00

CREATE TABLE

- Creation of a table using NOT NULL

CREATE TABLE Instructor

```
(InstID      VARCHAR(5),  
 InstName    VARCHAR(20) NOT NULL,  
 DeptName    VARCHAR(20),  
 Salary       DECIMAL(8,2),  
 PRIMARY KEY (InstID),  
 FOREIGN KEY(DeptName) REFERENCES Department(DeptName) ON DELETE SET NULL);
```

- NOT NULL is an Integrity constraint

- NOT NULL will ensure that each row always has an instructor name

DROP TABLE and ALTER TABLE

- To delete all rows in a table and the table schema

DROP TABLE Student;

- To delete all rows in a table, but retain the table schema

DELETE FROM Student;

- To add an attribute with assigned NULL values

ALTER TABLE Student **ADD** Shoesize DECIMAL(2,0);

- To drop an attribute

ALTER TABLE Student **DROP** Shoesize;

- To add a primary key or a foreign key

See page 42.

SQL Data Manipulation (book sec 3.9)

■ Modification of the Database

- Insert new rows into a table
- Delete rows from a table
- Update rows in a table

■ Examples

- Add new instructor, that is adding a row
- Delete an instructor, that is deleting a row
- Update salary for an instructor, that is updating a value in a row

INSERT

- Simplest form

```
INSERT INTO R (... ,Ai, ...) VALUES (... ,vali, ...);
```

- R represents a relation
- A_i is an attribute of R, val_i is a value expression

- Add a new row to Course

```
INSERT Course (CourseID, Title, DeptName, Credits)  
VALUES ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- Or equivalently

```
INSERT Course VALUES ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- Add a new row in Student with TotCredits set to NULL

```
INSERT Student VALUES ('3003', 'Green', '1993-04-16', 'Finance', NULL);
```

- Adding multiple rows

```
INSERT Course VALUES  
('CS-437', 'Database Systems', 'Comp. Sci.', 4),  
('CS-528', 'Big Data Systems', 'Comp. Sci.', 5),  
('CS-530', 'Data Warehouse', 'Comp. Sci.', 4);
```

- Can also take the form **INSERT INTO R (... ,A_i, ...) SELECT ... FROM ... WHERE ...;**

(See page 42 and demo example 2.8.)

DELETE

- Typical form

```
DELETE FROM R WHERE P;
```

- R represents a relation
- P is a (row) predicate over attribute names
- Removes those rows in R for which P is true

- Delete a row

- **DELETE FROM Instructor WHERE InstID = 45565;**

- Delete multiple rows

- **DELETE FROM Instructor WHERE DeptName='Finance';**

- Delete all rows

- **DELETE FROM Instructor;**

InstID	InstName	DeptName	Salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

UPDATE

■ Typical form

```
UPDATE R SET ..., Ai = vali, ... WHERE P;
```

- R represents a relation
- P is a (row) predicate over attribute names
- A_i is an attribute of R, val_i is a value expression
- changes the value of A_i to val_i for those rows for which P is true

■ Update multiple values in a single row

```
UPDATE Instructor SET Salary=85000, DeptName='Physics' WHERE InstID=45565;
```

■ Update multiple rows

- Increase salaries of instructors whose salary is over 80000 by 3%, and all others with a 5% raise.
- Write two update statements (the order is important):

```
UPDATE Instructor SET Salary=Salary*1.03 WHERE Salary>80000;
```

```
UPDATE Instructor SET Salary=Salary*1.05 WHERE Salary<=80000;
```

CASE Expressions

- Increase salaries of instructors whose salary is over 80,000 by 3%, and all others with a 5% raise.

```
UPDATE Instructor SET Salary =  
CASE  
WHEN Salary<=80000 THEN Salary*1.05  
ELSE Salary*1.03  
END;
```

- Consider giving more to those, who have plenty!

```
UPDATE Instructor SET Salary =  
CASE  
WHEN Salary BETWEEN 80000 AND 89999 THEN Salary+10000  
WHEN Salary BETWEEN 70000 AND 79999 THEN Salary+5000  
WHEN Salary BETWEEN 0 AND 69999 THEN Salary+2500  
ELSE Salary  
END;
```

SQL Data Queries (book sec 3.3-3.4, except 3.4.2) + (4.1.1 in 7th edition)

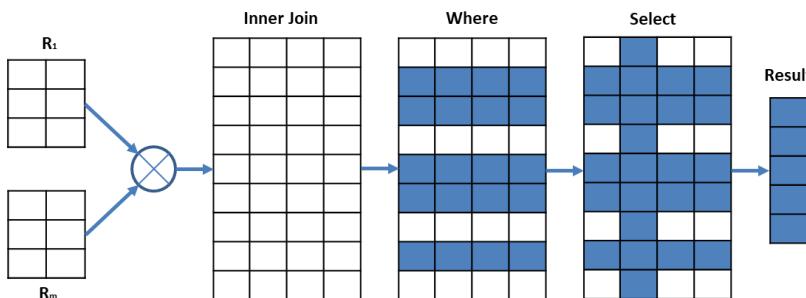
■ Basic Query Structure

- SQL Data Query Language provides the ability to query data
- The result of a SQL Query is a table

■ A typical basic SQL query has the form:

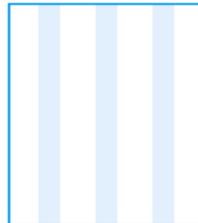
```
SELECT A1, A2, ..., An
FROM r1, r2, ..., rm
WHERE P;
```

- A_i represents an attribute
- r_i represents a relation
- P is a (row) predicate over attribute names. For each row it is either true or false.



SELECT

- The select clause lists the attributes desired in the result of a query
 - Corresponds to the projection operation of relational algebra



- Example: Like find the names of all instructors

```
SELECT InstName FROM Instructor;
```

InstID	InstName	DeptName	Salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

Note

- SQL names are case insensitive. You may use upper- or lower-case letters, for example InstName \equiv INSTNAME \equiv instname.

SELECT: DISTINCT, ALL

- SQL generally allows duplicate rows in tables as well as in query results.
- To force the elimination of duplicates, insert the keyword DISTINCT after SELECT.
 - Like find the names of all departments from the Instructor table and remove duplicates:

```
SELECT DISTINCT DeptName FROM Instructor;
```

- Keyword ALL specifies that duplicates are not removed

```
SELECT ALL DeptName FROM Instructor;
```

SELECT: using *, expressions, AS

- A '*' in the SELECT clause denotes “all attributes”

SELECT * FROM Instructor;

- The select clause can contain **arithmetic expressions** involving the operation, +, −, *, and /, and operating on constants or attributes of rows.
- The query:

SELECT InstID, Salary/12 AS Monthly FROM Instructor;

InstID	Monthly
10101	5416.666667
12121	7500.000000
15151	3333.333333

- Returns a table where the value of the attribute Salary is divided by 12, giving the monthly salary instead of the yearly salary.

SELECT: using built-in functions

- The select clause can contain *built-in functions* associated with the built-in datatypes.

- Example (for the Date datatype):

- `SELECT CURDATE();`

```
curdate0  
2015-06-22
```

- Example: Calculation of Age from Birth

- Basic rule:

Never store dynamic values like Age,
instead store static values like Birth,
and calculate dynamic values like Age.

```
SELECT StudName, Birth,  
TIMESTAMPDIFF(YEAR, Birth, CURDATE()) AS Age  
FROM Student;
```

StudName	Birth	Age
Zhang	1992-04-18	23
Shankar	1995-12-06	19
Brandt	1993-05-24	22
Chavez	1992-04-18	23
Peltier	1995-10-18	19
Levy	1995-08-01	19
Williams	1995-02-28	20
Sanchez	1995-06-04	20
Snow	1995-11-22	19
Brown	1994-03-05	21
Aoi	1993-09-18	21
Bourikas	1992-09-23	22
Tanaka	1992-06-02	23

WHERE

- WHERE clause specifies conditions for the rows to be included in the result



- **Example:**

Find all instructors in Comp. Sci. department with Salary > 70000

```
SELECT InstName FROM Instructor  
WHERE DeptName='Comp. Sci.' AND Salary>70000;
```

- Comparisons with <, <=, >, >=, =, <> can be applied to arithmetic expressions and strings.
- Comparison results can be combined using the logical connectives **AND, OR, and NOT**.

FROM (section 3.3.2)

- The **FROM** clause lists the tables involved in the query
 - FROM r**
 - FROM r₁, ..., r_m**
corresponds to **Cartesian Product** operation $r_1 \times \dots \times r_m$ of Relational Algebra, also called a **join**.
- Example: Find the Cartesian Product: Instructor x Teaches

Instructor(InstID, InstName, DeptName, Salary) Teaches(InstID, CourseID, SectionID, Semester, StudyYear)

SELECT * FROM Instructor, Teaches;

- Generates every possible Instructor-Teaches pair
 - The result table has as attributes the union of Instructor&Teaches attrs
- | InstID | InstName | DeptName | Salary | InstID | CourseID | SectionID | Semester | StudyYear |
|--------|----------|----------|--------|--------|----------|-----------|----------|-----------|
|--------|----------|----------|--------|--------|----------|-----------|----------|-----------|
- The result table has $n * m$ rows, where n is the number of rows in Instructor and m is the number of rows in Teaches.
 - Beware of data explosion! A Cartesian Product of A x B x C, where each table has 100 rows, results in 1.000.000 rows (either the memory area will overflow with an error, or the response time will go towards infinity)!
 - Cartesian Product is not very useful directly, but useful combined with the WHERE clause condition.

Cartesian Product: Instructor x Teaches

- Combines each row in Instructor with all rows in Teaches
- Instructor(InstID, InstName, DeptName, Salary) Teaches(InstID, CourseID, SectionID, Semester, StudyYear)

SELECT * FROM Instructor, Teaches LIMIT 14;

InstID	InstName	DeptName	Salary	InstID	CourseID	SectionID	Semester	StudyYear
10101	Srinivasan	Comp. Sci.	65000.00	76766	BIO-101	1	Summer	2009
12121	Wu	Finance	90000.00	76766	BIO-101	1	Summer	2009
15151	Mozart	Music	40000.00	76766	BIO-101	1	Summer	2009
22222	Einstein	Physics	95000.00	76766	BIO-101	1	Summer	2009
32343	El Said	History	60000.00	76766	BIO-101	1	Summer	2009
33456	Gold	Physics	87000.00	76766	BIO-101	1	Summer	2009
45565	Katz	Comp. Sci.	75000.00	76766	BIO-101	1	Summer	2009
58583	Califieri	History	62000.00	76766	BIO-101	1	Summer	2009
76543	Singh	Finance	80000.00	76766	BIO-101	1	Summer	2009
76766	Crick	Biology	72000.00	76766	BIO-101	1	Summer	2009
83821	Brandt	Comp. Sci.	92000.00	76766	BIO-101	1	Summer	2009
98345	Kim	Elec. Eng.	80000.00	76766	BIO-101	1	Summer	2009
10101	Srinivasan	Comp. Sci.	65000.00	76766	BIO-301	1	Summer	2010
12121	Wu	Finance	90000.00	76766	BIO-301	1	Summer	2010

SELECT COUNT(*) FROM Instructor, Teaches;

COUNT(*)
180

More JOIN Examples

- Given
 - Instructor(InstID, InstName, DeptName, Salary)
 - Teaches(InstID, CourseID, SectionID, Semester, StudyYear)
- For all instructors who have taught some course, find their names and the course ID of the courses they taught.

```
SELECT InstName, CourseID  
      FROM Instructor, Teaches  
     WHERE Instructor.InstID=Teaches.InstID;
```

InstName	CourseID
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

More JOIN Examples

- Given
 - Section(CourseID, SectionID, Semester, StudyYear, Building, Room, TimeSlotID)
 - Course(CourseID, Title, DeptName, Credits)
- Find the CourseID, Semester, StudyYear and Title of each course offered by the Comp. Sci. department

```
SELECT Section.CourseID, Semester, StudyYear, Title  
FROM Section, Course  
WHERE Section.CourseID=Course.CourseID AND DeptName='Comp. Sci.' ;
```

CourseID	Semester	StudyYear	Title
CS-101	Fall	2009	Intro. to Computer Science
CS-101	Spring	2010	Intro. to Computer Science
CS-190	Spring	2009	Game Design
CS-190	Spring	2009	Game Design
CS-315	Spring	2010	Robotics
CS-319	Spring	2010	Image Processing
CS-319	Spring	2010	Image Processing
CS-347	Fall	2009	Database System Concepts

NATURAL JOIN (section 3.3.3 in 6th edition = 4.1.1 in 7th edition)

■ NATURAL JOIN

- Matches rows with the same values for all common attributes, and retains only one copy of each common attribute value.
- **SELECT * FROM Instructor NATURAL JOIN Teaches;**

InstID	InstName	DeptName	Salary	CourseID	SectionID	Semester	StudyYear
10101	Srinivasan	Comp. Sci.	65000.00	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000.00	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000.00	CS-347	1	Fall	2009
12121	Wu	Finance	90000.00	FIN-201	1	Spring	2010
15151	Mozart	Music	40000.00	MU-199	1	Spring	2010
22222	Einstein	Physics	95000.00	PHY-101	1	Fall	2009
32343	El Said	History	60000.00	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000.00	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000.00	CS-319	1	Spring	2010
76766	Crick	Biology	72000.00	BIO-101	1	Summer	2009
76766	Crick	Biology	72000.00	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000.00	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000.00	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000.00	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000.00	EE-181	1	Spring	2009

Natural Join in this example is over the common attribute name InstID.

NATURAL JOIN versus JOIN

- For all instructors who have taught some course, find their names and the course ID of the courses they taught:

```
SELECT InstName, CourseID  
FROM Instructor, Teaches  
WHERE Instructor.InstID=Teaches.InstID;
```

- The above SELECT expression is equivalent with:

```
SELECT InstName, CourseID  
FROM Instructor NATURAL JOIN Teaches;
```

Renaming: AS in SELECT and FROM (section 3.4.1)

- The SQL allows renaming tables and attributes using the **AS** clause: OldName AS NewName
 - Example:

```
SELECT InstID, InstName, Salary/12 AS Monthly
FROM Instructor;
```
 - Example: Find the names of all instructors who have a higher salary than some instructor in the 'Comp. Sci.' department.

```
SELECT DISTINCT T.InstName
FROM Instructor AS T, Instructor AS S
WHERE T.Salary > S.Salary AND S.DeptName = 'Comp. Sci.');
```
- Keyword AS is optional and may be omitted
 $\text{Instructor AS T} \equiv \text{Instructor T}$

SELECT revisited: use of * (section 3.4.3)

- For all instructors who have taught some course, select all attributes.

```
SELECT Instructor.* FROM Instructor, Teaches  
WHERE Instructor.InstID=Teaches.InstID;
```

ORDER BY (section 3.4.4)

- List in alphabetic order the names of all instructors

```
SELECT DISTINCT InstName FROM Instructor ORDER BY InstName;
```

- We may specify **DESC** for descending order or **ASC** for ascending order, for each attribute; Ascending order is the default.
 - ... **ORDER BY InstName DESC;**

- Can sort on multiple attributes

```
SELECT DeptName, InstName  
FROM Instructor  
ORDER BY DeptName, InstName;
```

DeptName	InstName
Biology	Crick
Comp. Sci.	Brandt
Comp. Sci.	Katz
Comp. Sci.	Srinivasan
Elec. Eng.	Kim
Finance	Singh
Finance	Wu
History	Califieri
History	El Said
Music	Mozart
Physics	Einstein
Physics	Gold

Making New Tables from Old Tables

- Create a new table, like the old table, and insert values

```
CREATE TABLE Specialist1 LIKE Instructor; # Create an empty table  
INSERT Specialist1 SELECT * FROM Instructor WHERE DeptName='Comp. Sci.';
```

- Create a table with contents from an old table

```
CREATE TABLE Specialist2 SELECT * FROM Instructor WHERE DeptName='Physics';
```

```
SELECT * FROM Specialist1;
```

InstID	InstName	DeptName	Salary
10101	Srinivasan	Comp. Sci.	65000.00
45565	Katz	Comp. Sci.	75000.00
83821	Brandt	Comp. Sci.	92000.00

```
SELECT * FROM Specialist2;
```

InstID	InstName	DeptName	Salary
22222	Einstein	Physics	95000.00
33456	Gold	Physics	87000.00

- However, Specialist1 has a Primary Key(InstID), while Specialist2 has none!

```
ALTER TABLE Specialist2 ADD PRIMARY KEY(InstID); # To define Primary Key
```

- Neither Specialist1 nor Specialist2 has Foreign Keys defined!

```
ALTER TABLE Specialist1 ADD FOREIGN KEY(DeptName)
```

```
REFERENCES Department(DeptName); # To define a Foreign Key
```



Summary

- University Database Example
- Data Definition: CREATE TABLE, DROP TABLE and ALTER TABLE.
- Data Manipulation: INSERT, UPDATE and DELETE data
- Data Queries: SELECT ... FROM ... WHERE ...

Readings

- In Database Systems Concepts (6th and 7th edition) please read sections 3.1, 3.2, 3.9 (except parts using subqueries), 3.3-3.4 (except 3.4.2). In 7th edition also read 4.1.1.
- Pay special attention to the Summary

Next time

- The remaining parts of chapter 3: more advanced queries

Tools Installations

- *If you have not yet made exercises 1-2 of slide set no 1.5 from the first week then do them before starting on the exercises of this slide set.*

Demo Exercises



Demo Exercises clarify ideas and concepts from the Lecture to provide you with good Database Skills.

Discuss and redo the Demo Exercises.

Create and Download the University Database

Create the University Database

Start MySQL Workbench. Once it is running select connect to database and login with the Username and Password you created during installation.

Define a name for a university database (e.g. University) and start to use it by executing the following SQL statements:

```
CREATE DATABASE University;  
USE University;
```

Download the University Database

To create the university tables, download the “UniversityDB.sql” file to your PC from the *Databases* folder under DTU Learn Content.

Use 'Open an SQL script file in a new query tab' in MySQL Workbench, and select the file 'UniversityDB.sql' and execute it.

You can check all the relations and their instances by using queries like:

```
DESCRIBE Student;  
SELECT * FROM Student;
```

Basic SQL Queries

Demo Examples

The following examples will provide you with some basic SQL query knowledge, that you will need to solve the exercises. It is optional, but highly recommended, to copy the Demo Example queries into the workbench, and run them on the university database. Try to change some of the conditions in the queries, this will help you to get familiarized with basic SQL queries.

Each Demo Example contains a plain text, describing the query, the actual SQL query and the resulting dataset.

Please be aware that the examples given are not the only way to write the SQL queries, but just one way to do it. Often you will be able to write queries in more than one way.

2.1 SELECT all attributes

Get all the departments of the university.

```
SELECT * FROM Department;
```

DeptName	Building	Budget
Biology	Watson	90000.00
Comp. Sci.	Taylor	100000.00
Elec. Eng.	Taylor	85000.00
Finance	Painter	120000.00
History	Painter	50000.00
Music	Packard	80000.00
Physics	Watson	70000.00

2.2 SELECT some attributes

Get the name and building of all departments of the university.

```
SELECT DeptName, Building FROM Department;
```

DeptName	Building
Biology	Watson
Comp. Sci.	Taylor
Elec. Eng.	Taylor
Finance	Painter
History	Painter

Basic SQL Queries

2.3 SELECT rows based on single condition

Get all departments with a budget greater than or equal to 100000.

```
SELECT * FROM Department  
WHERE Budget >= 100000;
```

DeptName	Building	Budget
Comp. Sci.	Taylor	100000.00
Finance	Painter	120000.00

2.4 SELECT rows based on multiple conditions

Get all departments with a budget greater than or equal to 70000 and that is located in the Taylor building.

```
SELECT * FROM Department  
WHERE Budget >= 70000  
AND Building = "Taylor";
```

DeptName	Building	Budget
Comp. Sci.	Taylor	100000.00
Elec. Eng.	Taylor	85000.00

2.5 JOIN with condition and NATURAL JOIN

Get the instructors (id and name) that work in a department with a budget ≥ 100000 .

```
SELECT InstID, InstName  
FROM Instructor, Department  
WHERE  
Instructor.DeptName = Department.DeptName  
AND Budget >= 100000;
```

```
SELECT InstID, InstName  
FROM Instructor NATURAL JOIN Department  
WHERE Budget >= 100000;
```

InstID	InstName
10101	Srinivasan
45565	Katz
83821	Brandt
12121	Wu
76543	Singh

(The order of rows might differ for different systems.)

SQL Data Manipulation

2.6 UPDATE with calculation

Increase the salary of each instructor in the Comp. Sci. department by 10%.

```
UPDATE Instructor SET Salary = Salary * 1.10  
WHERE DeptName = 'Comp. Sci.';
```

2.7 DELETE

Delete all courses with credits less than or equal to 3.

```
DELETE FROM Course  
WHERE Credits <= 3;
```

Important note: In order to be able to execute the statement above, you will need to disable *safe mode* (if this is not already done) by executing:

```
SET SQL_SAFE_UPDATES = 0;
```

The reason is that if `SQL_SAFE_UPDATES = 1`, then the `WHERE` clause (of delete and update statements) must refer to the key attribute, but that is not the case in the delete statement above.

If you later wish to enable the safe mode again, this can be done by executing:

```
SET SQL_SAFE_UPDATES = 1;
```

2.8 INSERT with SELECT

Insert every student whose `TotCreds` attribute is greater than 100 as an instructor in the same department, with a salary of 10000.

```
INSERT INTO Instructor  
SELECT StudID, StudName, DeptName, 10000  
FROM Student  
WHERE TotCredits > 100;
```

PS. Run the `UniversityDB.sql` script again to restore tables to the original instance.

2.9 INSERT with VALUES: see 2.11

SQL Data Definition and Data Manipulation

2.10 Create table GradePoints(Grade, Points)

to provide a conversion from letter grades in the Takes table to numeric scores; For example an “A” grade could be specified to corresponds to 4 points, an “A-“ to 3.7 points, a “B+” to 3.3 points and so on.

```
CREATE TABLE GradePoints (
    Grade VARCHAR(2) PRIMARY KEY,
    Points DECIMAL(3,1));
```

2.11 Populate the table with data

```
INSERT GradePoints VALUES ('A', 4.0);
INSERT GradePoints VALUES ('A-', 3.7);
INSERT GradePoints VALUES ('B+', 3.3);
INSERT GradePoints VALUES ('B', 3.0);
INSERT GradePoints VALUES ('B-', 2.7);
INSERT GradePoints VALUES ('C+', 2.3);
INSERT GradePoints VALUES ('C', 2.0);
INSERT GradePoints VALUES ('F', 0.0);
```

2.12 Drop table

```
DROP TABLE GradePoints;
```

Exercises



Please answer all exercises
to demonstrate your
Database Skills.

Solutions are available at 11:45

Basic SQL Queries

Exercises

SQL queries should be run against the University database.

Have a print of the University Database Schema Diagram and the Database Instance readily available.

2.13 SELECT all attributes

Get all data in the Student table

2.14 SELECT all attributes

Get all data in the Course table

2.15 SELECT only one attribute

Get the name of all students

2.16 SELECT multiple attributes

Get the name and total credits of all students

2.17 SELECT multiple attributes

Get the name, salary and department of all instructors.

2.18 SELECT only some rows

Get the names of all students with a total credit of more than 100.

2.19 SELECT rows based on multiple conditions

Get the students in Computer Science with a total credit of more than 100.

2.20 SELECT rows based on multiple conditions

Get the rooms with a capacity between 25 and 50, or located in the Painter building.

2.21 SELECT rows based on single condition

Get all department names not located in the Taylor building.

2.22 SELECT with two tables

What are the Course ID, year and grade for all courses taken by student Shankar?

SQL Data Manipulation

2.23 INSERT with multiple rows

Create two new Comp. Sci. courses CS-102 and CS-103 in table Course titled Weekly Seminar and Monthly Seminar, both with 0 credits.

2.24 INSERT with multiple NULL values

Create a section for both CS-102 and CS-103 in Fall 2009, both with SectionID 1.

2.25 INSERT with SELECT and NULL

In table Takes enroll every student in the Comp. Sci. department in the section for CS-102.

2.26 DELETE

Delete both courses CS-102 and CS-103 in the Takes table.

2.27 UPDATE

Move the Finance department to the Taylor building.

PS. Run the UniversityDB Script to restore tables to initial instances.

Create a Database & Populate it with Data

2.28 Create a Database

Write SQL DDL statements corresponding to the Relation Schemas below for an Insurance Database.

Person (DriverID, DriverName, Address)

Car (License, Model, ProdYear)

Accident (ReportNumber, AccDate, Location)

Owns (DriverID, License)

Participants (ReportNumber, License, DriverID, DamageAmount)

Make any reasonable assumptions about data types, and declare primary and foreign keys.

2.29 Populate a Database

Write SQL DML statements to populate the database with data, to end up with:

SELECT * FROM Person;

DriverID	DriverName	Address
31262549	Hans Hansen	Jernbane Alle 74, 2720 Vanløse

SELECT * FROM Car;

License	Model	ProdYear
JW46898	Honda Accord Aut. 2.0	2001

SELECT * FROM Accident;

ReportNumber	AccDate	Location
3004000121	2015-06-18	2605 Brøndby

SELECT * FROM Owns;

DriverID	License
31262549	JW46898

SELECT * FROM Participants;

ReportNumber	License	DriverID	DamageAmount
3004000121	JW46898	31262549	6800