

SQL DDL, Data Modification

ISYS2120 Data and Information Management

Prof Alan Fekete

University of Sydney

Acknowledge: slides from Uwe Roehm and Alan Fekete, and from the materials associated with reference books (c) McGraw-Hill, Pearson

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**). The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Agenda

- CREATE TABLE syntax
- Constraints
 - NOT NULL
 - PRIMARY KEY
 - Also for composite primary key
 - UNIQUE
 - FOREIGN KEY
 - ON INSERT, ON DELETE choices
 - CHECK
- DROP TABLE
- Data Modification
 - INSERT
 - UPDATE
 - DELETE

Schema definition

- A relational dbms knows the logical schema of the database it manages
 - Indeed, data cannot be stored except when the schema is already known
 - This is in contrast to some other platforms (eg JSON stores), which can accept data and then be told (or even figure out) the schema
- The schema is described in SQL statements (part of Data Definition Language, also abbreviated as DDL)
 - Give names of tables, names of columns, datatypes of columns, and also various constraints
- There are also commands that modify the schema
- Schema descriptions are stored in the system catalogue, which can be queried
 - Eg a table which gives information about the attributes of all tables

Creating Tables in SQL

- Creation of tables:

CREATE TABLE *name* (*list-of-columns-with-types*)

- Example: Create the Instructor relation.

Observe that the datatype (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Instructor (lname VARCHAR(20) ,  
                           fname  VARCHAR(20) ,  
                           salary  INTEGER ,  
                           birth   DATE ,  
                           hired   DATE ) ;
```

- Remember that table and column names are SQL identifiers (and case-insensitive)
- Remember not to use any keyword as an identifier (eg, don't try to have a column called `date`)
- After performing this, the table exists, but it starts empty (contains no rows yet)

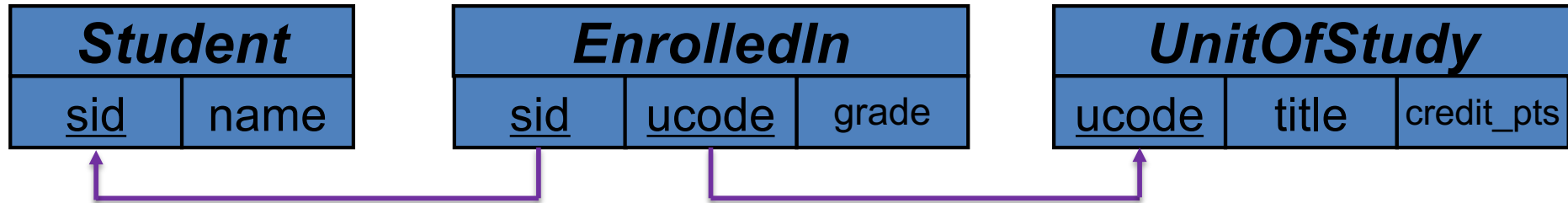
Base Datatypes of SQL

Full list is non-examinable, except for the types we introduce elsewhere, in slides or Ed lessons

Base Datatypes	Description	Example Values
SMALLINT INTEGER BIGINT	Integer values	1704, 4070
DECIMAL(p,q) NUMERIC(p,q)	Fixed-point numbers with precision p and q decimal places	1003.44, 160139.9
FLOAT(p) REAL DOUBLE PRECISION	floating point numbers with precision p	1.5E-4, 10E20
CHAR(q) VARCHAR(q) CLOB(q)	alphanumeric character string types of fixed size q respectively of variable length of up to q chars	'The quick brown fox jumps...', 'ISYS2120'
BLOB(r)	binary string of size r	B'01101', X'9E'
DATE	date	DATE '1997-06-19', DATE '2001-08-23'
TIME	time	TIME '20:30:45', TIME '00:15:30'
TIMESTAMP	timestamp	TIMESTAMP '2002-08-23 14:15:00'
INTERVAL	time interval	INTERVAL '11:15' HOUR TO MINUTE

(from Türker, ORDBMS 2004/2005)

Create Table Example



Note: in SQL DDL, layout on lines is not significant! Note: this relational schema is changed from previous lecture

/* We will add more for constraints, later */

```
CREATE TABLE Student (  
    sid    INTEGER,  
    name VARCHAR(20)
```

```
);
```

```
CREATE TABLE UnitOfStudy
```

```
(
```

```
    ucode CHAR(8),  
    title VARCHAR(30),  
    credit_pts INTEGER
```

```
);
```

```
CREATE TABLE EnrolledIn (  
    sid INTEGER, ucode CHAR(8), grade INTEGER
```

```
);
```

SQL Schema Names

- DBMS Servers are typically shared by multiple data owners
 - It would be very difficult to ensure that everyone uses different names for tables, constraints, etc
 - So, we want a separate name-space for each data owner
 - SQL allows this: one can have several different schema stored in the dbms
 - Full name of a table is then `schemaname.tablename`
 - **CREATE SCHEMA** ... command
 - See E.g. <https://www.postgresql.org/docs/15/sql-createschema.html>
 - If not stated, implicitly, the schema any command refers to, is the one named by user name of user who issues the SQL command
- See later weeks for discussion of access control on references to another user's schema

Integrity Constraints

- **Integrity Constraint (IC):** a condition that must be true for *any* contents of the database; e.g., domain constraints, uniqueness constraints and more complicated ones too.
 - ICs can be declared in the schema
 - They are specified when schema is defined.
 - All declared ICs are checked whenever relations are modified.
- A *legal* instance of a relation is one where the contents satisfies every specified IC.
 - When ICs are declared in the schema, DBMS will not allow illegal instances to arise.
 - So stored data is more faithful to real-world meaning.
 - This can help avoid some data entry errors, too!
 - Not examinable comment: Also, DBMS can sometimes use its knowledge of ICs to improve performance
- Having ICs declared explicitly in schema, is a valuable aspect of having the DBMS *manage* the data (not just store it)

The Special NULL “Value”

- SQL-based RDBMS allows a special entry **NULL** in a column to represent facts that are not applicable, or not yet known
 - This can occur in a column of any datatype (unless schema explicitly prevents it)
 - This is different than the VARCHAR 'Null' or the empty string ''
 - Warning: When displaying contents of a table, a cell with NULL is often shown empty!
- ▶ Eg a new employee has not yet been allocated to a department
- ▶ Eg salary, hired may not be meaningful for adjunct lecturers

Eg INSTRUCTOR table in a university

lname	fname	salary	birth	hired
Jones	Peter	35000	1970	1998
Smith	Susan	null	1983	null
Smith	Alan	35000	1975	2000

Not examinable comment: NULL in SQL is similar to None in Python, null in Java etc

Tradeoffs of use of NULL

- Is NULL a helpful way to deal with an unknown or not applicable attribute?
 - Instead of eg using a special value which would not make ordinary sense
 - Eg one might use age = -1 to represent that age is not known
- Advantage of using NULL:
NULL is useful because system knows to treat it specially
 - Eg calculations like avg(age) or min(age) ignore NULL cases, but will give incorrect answer if some values are -1
- Disadvantage:
NULL causes complications in the definition of many operations (eg is 27 < NULL? Is 27 > NULL? Is NULL = NULL?)
 - These complexities can confuse stakeholders, and sometimes this leads to incorrect query-writing
 - We describe SQL approach for this in Lecture 4B

NOT NULL Constraint

- One domain constraint is to insist that no value in a given column can be null
 - The value of that column in a row can't be unknown; the concept can't be inapplicable
- In SQL, we can declare this: append NOT NULL to the field declaration

```
CREATE TABLE Instructor (  
    lname VARCHAR(20) NOT NULL,  
    fname  VARCHAR(20) NOT NULL,  
    salary INTEGER,  
    birth  DATE NOT NULL,  
    hired  DATE );
```

Primary Key constraint

- Recall that many entities have identifiers, which can be used to distinguish one entity from another in the entity set, even when many attributes are the same
 - Often, an identifier is completely artificial, and the value has no meaning at all
 - Sometimes, the identifier is a sequence number
- In a relational schema, we can declare that a column is a Primary Key (often abbreviated as PK in textbooks or diagrams)
 - This allows it to be used as identifier in other tables, to connect information
 - It will enforce that every row has a different value for this column
 - It will enforce that no row has NULL as value for this column

SQL ways to indicate a PK

- Append to end of column definition

```
CREATE TABLE Student (  
  sid    INTEGER PRIMARY  
        KEY,  
  name VARCHAR(20)  
);
```

- Extra clause in table creation

```
CREATE TABLE Student (  
  sid    INTEGER,  
  name VARCHAR(20),  
        PRIMARY KEY (sid)  
);
```

- Named constraint in table creation

```
CREATE TABLE Student (  
  sid    INTEGER,  
  name VARCHAR(20),  
        CONSTRAINT Student_PK  
        PRIMARY KEY (sid)  
);
```

- Separate named constraint added later

```
CREATE TABLE Student (  
  sid    INTEGER,  
  name VARCHAR(20)  
);  
ALTER TABLE Student (  
  ADD CONSTRAINT Student_PK  
  PRIMARY KEY (sid)  
);
```

Composite PK

- In some tables, no single column is an identifier, but a combination of columns may be able to distinguish the rows
- Eg, maybe for airplanes, each manufacturer has its own set of serial numbers,
 - it is possible to have two rows with the same serialno, when they have different manufacturer
 - And it is possible to have two rows with the same manufacturer, when they have different serialno
 - But the combination (manufacturer, serialno) is suitable as an identifier
 - This is called a composite primary key

```
CREATE TABLE Airplane (  
  manufacturer VARCHAR(20),  
  serialno INTEGER,  
  weight REAL,  
  PRIMARY KEY (manufacturer, serialno)  
);
```

UNIQUE constraint

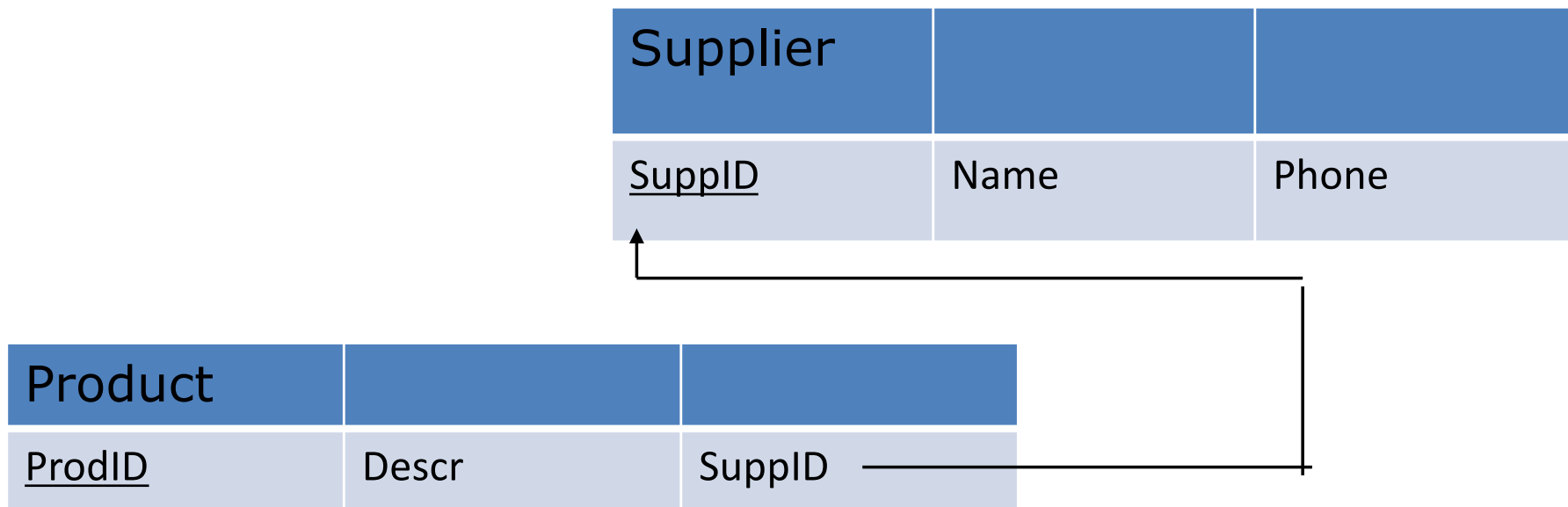
- A table can have at most one primary key declared (this PK may be a single column, or a combination of columns)
- Sometimes, there are other columns or combinations that are necessarily different among rows
 - We call any combination of columns which can distinguish the rows, as a “candidate key”
- SQL allows multiple constraints on a table, each saying a column or combination of columns is UNIQUE
- What about NULL values in a UNIQUE column?
 - SQL standard allows even more than one row with NULL value in a UNIQUE column (unless the column was declared NOT NULL)
 - However, many platforms limit to having at most one row with NULL in that column
 - PostgreSQL default does not allow multiple NULL entries in a UNIQUE column; but PostgreSQL (since version 15) has syntax to change this

Foreign Key

- When we use values to connect information across tables, we expect that there will be something with that value in the referenced table
- We can make this explicit with a FOREIGN KEY constraint (often abbreviated FK in text or diagrams)
 - In the SQL standard, the reference must be to the declared primary key of the referenced table
 - Some platforms (including PostgreSQL) allow reference to candidate key instead
- The value in a foreign key column is allowed to be NULL
 - Unless the column is separately declared as NOT NULL

Foreign Key Example

- For the relational schema diagram
- Product(SuppID) is a foreign key referencing SuppID, the primary key in Supplier
- Any value in the SuppID column in Product, is either a value in SuppID in Supplier, or else it is NULL



Foreign Key Constraint in SQL


- Several ways to express this, among them

```
CREATE TABLE Product ( ProdID INTEGER,  
    descr VARCHAR(10),  
    SuppID INTEGER REFERENCES Supplier(SuppID));
```

Or

```
CREATE TABLE Product ( ProdID INTEGER,  
    descr VARCHAR(10),  
    SuppID INTEGER REFERENCES Supplier);
```

When no column is mentioned for referenced table, implicit reference is to its primary key



Or

```
CREATE TABLE Product ( ProdID INTEGER,  
    descr VARCHAR(10),  
    SuppID INTEGER,  
    FOREIGN KEY (SuppID) REFERENCES Supplier(SuppID));
```

Or

```
CREATE TABLE Product ( ProdID INTEGER,  
    descr VARCHAR(10),  
    SuppID INTEGER,  
    CONSTRAINT Product_FK FOREIGN KEY (SuppID) REFERENCES  
    Supplier(SuppID));
```

Note that when referenced table primary key is composite (multiple columns), and therefore also referencing table needs a similar combination as the foreign key, then only the last two of these formats can be used

Example: Foreign Key in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.

```
CREATE TABLE Enrolled
(   sid CHAR(10),   uos CHAR(8),   grade CHAR(2),
    PRIMARY KEY (sid,uos),
    FOREIGN KEY (sid) REFERENCES Student )
```

Student

sid	name	age	country
53666	Jones	19	AUS
53650	Smith	21	AUS
54541	Ha Tsch	20	CHN
54672	Loman	20	AUS

Enrolled

sid	uos	grade
53666	COMP5138	CR
53666	INFO4990	CR
53650	COMP5138	P
53666	SOFT4200	D
54221	INFO4990	F

??? Dangling reference

This state is NOT allowed by constraint

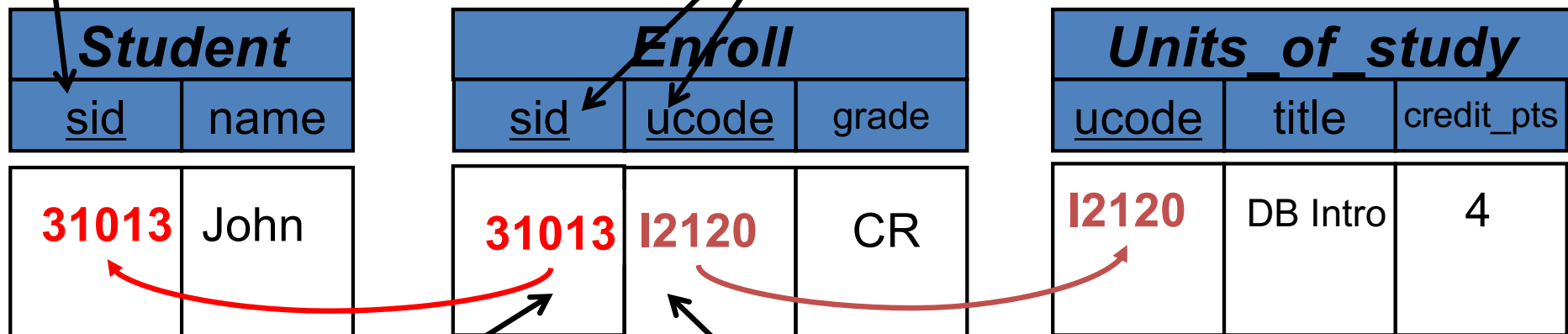
Referential Integrity

- The term “referential integrity” is often used for a system which ensures foreign key constraints
- The value in a referring column must be able to match a row in the referenced table
 - So, there are no “dangling references”
- Some books expand the term to include constraints where the reference may be to a candidate key, not only to the primary key of the referenced table

Summary of terms

Primary key identifies each tuple of a relation.

Composite Primary Key consisting of more than one attribute.



Foreign key is a (set of) attribute(s) in one relation that `refers' to a tuple in another relation by giving the value of its primary key (like a `logical pointer').

Another foreign key

Create Table Example with PKs/FKs

<i>Student</i>	
<u>sid</u>	name

<i>Enrolled</i>		
<u>sid</u>	<u>ucode</u>	grade

<i>Unit_of_Study</i>		
<u>ucode</u>	title	credit_pts

```
CREATE TABLE Student ( sid INTEGER, ... ,  
    CONSTRAINT Student_PK PRIMARY KEY (sid)  
);
```

```
CREATE TABLE UoS ( ucode CHAR(8), ... ,  
    CONSTRAINT UoS_PK PRIMARY KEY (ucode)  
);
```

```
CREATE TABLE Enrolled ( sid INTEGER, ucode CHAR(8), grade CHAR(2),  
    CONSTRAINT Enrolled_FK1 FOREIGN KEY (sid) REFERENCES Student,  
    CONSTRAINT Enrolled_FK2 FOREIGN KEY (ucode) REFERENCES UoS,  
    CONSTRAINT Enrolled_PK PRIMARY KEY (sid,ucode)  
);
```

Choosing the Correct Key Constraints

- Careful: Used carelessly, an IC can prevent the storage of database instances that arise in practice!
- Example:
Attempt to model that a student can get only a single grade per course.

<pre>CREATE TABLE Enrolled (sid INTEGER, cid CHAR(8), grade CHAR(2), PRIMARY KEY (sid,cid))</pre>	vs.	<pre>CREATE TABLE Enrolled (sid INTEGER, cid CHAR(8), grade CHAR(2), PRIMARY KEY (sid, cid), UNIQUE (sid, grade))</pre>
--	-----	--

- “For a given student and course, there is a single grade;
the same grade can be achieved
by several students in a course.”
- “For a given student and course,
there is a single grade;
and a student can achieve *a certain grade only once*.” [This is probably
not what is intended]

Brainteaser

- Given the following example a table

```
CREATE TABLE Test (  
    a INTEGER,  
    b INTEGER UNIQUE,  
    PRIMARY KEY (a,b)  
);
```

- Would this be a legal database instance?

a	b
1	1
1	2
1	3
2	1
2	4

Summary: Keys and NULLs

- PRIMARY KEY
 - At most one primary key declared for a table
 - The rows must have distinct values in this column(s) and NULL values are not allowed in this column(s)
- UNIQUE (candidate key)
 - Possibly many *candidate keys* (specified using UNIQUE)
 - According to the ANSI standards SQL:92, SQL:1999, and SQL:2003, a UNIQUE constraint should disallow duplicate non-NULL values, but allow multiple NULL values.
 - Many DBMS (e.g. Oracle or SQL Server) implement only a crippled version of this, allowing a single NULL but disallowing multiple NULL values....
- FOREIGN KEY
 - By default allows nulls
 - If there must always be a referenced row, then must combine with NOT NULL constraint

CHECK Constraints

- In SQL the domain of a column is given as a simple datatype
- Sometimes, we know more about the valid values that can be there
 - Eg Salary should be positive
 - Eg UoSCode should be 4 letters then 4 digits
- We can define a Boolean property that is required to always be true for every row

```
CREATE TABLE Employee
( empID    INTEGER,
  ...
  salary   INTEGER CHECK (salary > 0), ... );
```

- If the property involves several columns, it needs to be a separate clause in the SQL, not just appended to the column declaration

Changing table definition in SQL

- Deletion of a table:

DROP TABLE *name*

- the schema information and the tuples in the table, are all removed.
- Example: Destroy the Instructor relation

DROP TABLE Instructor

- Existing tables can have their structure changed

ALTER TABLE *name* **ADD COLUMN ... | ADD CONSTRAINT... | ...**

- Huge variety of vendor-specific options; cf. online documentation

Modifying table content using SQL DML

- Addition of extra tuples into a table: INSERT command
- Modification of existing tuples in a table: UPDATE ... WHERE command
- Removal of existing tuples from a table: DELETE ... WHERE command

INSERT

- Addition of extra tuples into a table
- INSERT one tuple with given values
 - Simplest form is

INSERT INTO *table* **VALUES** (*list-of-expression*)

- This expands the table with an extra row, containing exactly the entries given, in that order
- Eg to insert a row in Student table, with sid = 12345678, and name = 'Smit' (since the columns of Student are (sid,name))

```
INSERT INTO Student VALUES (12345678, 'Smit')
```

- Some platforms allow multiple rows to be inserted in one command

```
INSERT INTO Student VALUES (12345678, 'Smit'), (12444455, 'Cheng')
```

- You can name the columns to correspond to the entries you provide (in this case, other columns not listed will get their default value). Eg if default for name is NULL

```
INSERT INTO Student (sid) VALUES (12345678)
```

- You can add all the rows that are the the result of a query

```
INSERT INTO Student SELECT ...
```

- This will find the result of the SELECT command, and then add them all as rows to Student table

UPDATE

- Modification of existing tuples in a table
- Provide a WHERE clause to identify which rows of the table will be modified
 - Note: the evaluation uses the previous values of columns of each row
- For each entry which you want to modify, in SET clause, give an expression which calculates the new value to appear in this entry
- Simplest case: new entry is a constant

UPDATE UoS

SET title = 'Awesome Data'
WHERE ucode = 'ISYS2120'

- Also simple, new entry is an expression formed using existing values of columns in that row

UPDATE UoS

SET credit_pts = 2*credit_pts
WHERE ucode = 'ISYS2120'

- More complex, use a correlated subquery to calculate the new value (this can use data from other tables, as well as from the row being modified)
- **General Syntax:**

UPDATE table **SET** column=expression {,column=expression}
[**WHERE** search_condition]

WHERE clause

Just as in SELECT statement

DELETE

- Removal of existing tuples from a table
- Provide a WHERE clause to identify which rows of the table will be removed
 - Note: the evaluation uses the previous values of columns of each row

- **General Syntax:**

DELETE FROM *table* [**WHERE** *search_condition*]

- Example:

DELETE FROM Student WHERE name LIKE '%Fekete'

WHERE clause

Just as in SELECT statement

Referential Integrity on Data Modification

- Consider Student and Enrolled;
sid in Enrolled is a foreign key that references Student.
- What should be done if an Enrolled tuple with a non-existent student *sid* is inserted? (Answer: *We reject the attempted insertion!*)
- What should be done if a Student tuple is deleted? Choices:
 - Also delete all Enrolled tuples that refer to it.
 - Disallow deletion of a Student tuple that is referred to.
 - Set *sid* in Enrolled tuples that refer to it to a *default sid*.
 - Set *sid* in Enrolled tuples that refer to it to a special value *null*, denoting '*unknown*' or '*inapplicable*'.
- Similar if primary key of Student tuple is updated, this could leave some Enrolled tuples without anything to reference!
 - Choices of whether to allow the change, and if so, how to treat the Enrolled tuples

Referential Integrity in SQL

- SQL/92, SQL:1999 and SQL:2003 support all 4 options on deletes and (separately) on updates.
 - Default is **NO ACTION** (*delete/update is rejected*)
 - **CASCADE** (also delete all tuples that refer to deleted tuple)
 - **SET NULL / SET DEFAULT** (sets foreign key value of referencing tuple)

UnitOfStudy

<u>uosCode</u>	title	credit_pts	taughtBy
----------------	-------	------------	----------

Example has different schema than previously in lecture

```
CREATE TABLE UnitOfStudy
( uosCode    CHAR(8) ,
  title      VARCHAR(80) ,
  credit_pts  INTEGER,
  taughtBy   INTEGER DEFAULT 1,
  PRIMARY KEY (uosCode) ,
  FOREIGN KEY (taughtBy)
  REFERENCES Professor
  ON UPDATE CASCADE
  ON DELETE SET DEFAULT )
```

Note: the clause is stated in the *referring* table, and says what to do to it, on changes to the *referenced* table

Professor

<u>empid</u>	name
--------------	------

References

- Silberschatz/Korth/Sudarshan(7ed)
 - Chapter 2.1 - 2.4; Chapter 3.2, 3.7.1, and 3.9; Chapter 4.4 and 4.5

Also

- Kifer/Bernstein/Lewis(complete version, 2ed)
 - Chapter 3.3
- Ramakrishnam/Gehrke(3ed)
 - Chapter 3.1-3.2 and 3.7
 - *3.7 includes views, which we cover later*
- Garcia-Molina/Ullman/Widom(complete book, 2ed)
 - Chapter 2.3, Chapter 7.1

Summary

- CREATE TABLE syntax
- Constraints
 - NOT NULL
 - PRIMARY KEY
 - Also for composite primary key
 - UNIQUE
 - FOREIGN KEY
 - ON INSERT, ON DELETE choices
 - CHECK
- DROP TABLE
- Data Modification
 - INSERT
 - UPDATE
 - DELETE