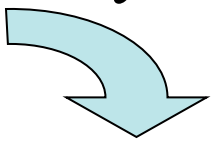
Relational Model 2

Professor Jessica Lin

…how do we create an actual database on our computers?

We use SQL, a language that allows us to build, modify and query databases.

instructor(<u>*ID*</u> : string, *name* : string, salary: real)



```
SQL> CREATE TABLE instructor
  2  (ID CHAR(5),
  3  name VARCHAR(50),
  4  salary integer,
  5  PRIMARY KEY(ID));

Table created.
```

SQL (Structured Query Language)

- SQL is a language that allows us to build, modify and query databases.
- SQL is an ANSI standard language. American National Standards Institute
- SQL is the “engine” behind Oracle, Microsoft SQL Server, etc.
- Most of these systems have built GUIs on top of the command line interface, so you don’t normally write statements directly in SQL (although you can).

Data Definition Language

The SQL **data-definition language (DDL)** allows the specification of information about relations, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- Also other information such as
  - The set of indices to be maintained for each relations.
  - Security and authorization information for each relation.
  - The physical storage structure of each relation on disk.

Domain Types in SQL

- **char(n).** Fixed length character string, with user-specified length *n*.
- **varchar(n).** Variable length character strings, with user-specified maximum length *n*.
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with *n* digits to the right of decimal point.
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.
- More are covered in Chapter 4.

Creating Relations in SQL

- Creates a *student* relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
SQL> CREATE TABLE student
    (ID CHAR(5),
     name VARCHAR(50),
     tot_cred integer,
     PRIMARY KEY(ID));
```

In Oracle, use VARCHAR2 instead of VARCHAR.

- As another example, the *advisor* table holds information about the advising relationship between instructors and students.

```
SQL> CREATE TABLE advisor
    (s_ID CHAR(5),
     i_ID CHAR(5),
     PRIMARY KEY(s_ID));
```

Creating Domains

- Say you want to restrict the values of tot_cred (0 <= tot_cred < 300)

- Approach 1: Specify constraint when defining the table

```
CREATE TABLE student
        (ID CHAR(5),
         name VARCHAR2(50),
         tot_cred INTEGER CHECK(tot_cred < 300 AND tot_cred >= 0));
```

Creating Domains

- Approach 2: After CREATING TABLE, use ALTER TABLE

```
CREATE TABLE student
   (ID CHAR(5),
    name VARCHAR(50),
    tot_cred integer);
```

```
ALTER TABLE Students
ADD CONSTRAINT check_cred CHECK(tot_cred < 300 AND tot_cred >= 0)
```

To specify a set of allowed values, do something like this (using either approach):

… CHECK(gender='M' OR gender='F')

Getting Info About Existing Tables

To get the list of existing tables in your database:

```
SELECT table_name
FROM user_tables;
```

or

```
SELECT table_name
FROM all_tables
WHERE owner='YOUR_ACCOUNT_NAME_IN_UPPER_CASE';
```

To get more about an existing table, say, student:

```
Describe student;
```

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO  student (ID, name, tot_cred)
VALUES (11111, 'Smith', 12);
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM student
WHERE name = 'Smith';
```

The SQL Query Language

- To find all 18 year old students, we can write:

```
SELECT  *
FROM  student S
WHERE  s.ID = 99001;
```

ID	***name***	***tot_cred***
98988	Tanaka	60
99001	Smith	51
90021	Jackson	12
95012	Lincoln	96

- To show just ID and name columns, replace the first line with:

```
SELECT  S.ID, S.name
```

Querying Multiple Relations

- What does this query compute?

```
SELECT  S.name, A.i_ID
FROM  student S, advisor A
WHERE  S.name = 'Smith' AND S.ID = A.s_ID;
```

student

ID	*name*	*tot_cred*
98988	Tanaka	60
99001	Smith	51
90021	Jackson	12
95012	Lincoln	96

advisor

i_ID	*s_ID*
10101	98988
10101	99001
12121	98988

we get:

S.name	A.i_ID
Smith	10101

Destroying and Altering Relations

- Destroys the relation Students. The schema information *and* the tuples are deleted.

```
DROP TABLE student;
```

- The schema of "student" is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

```
ALTER TABLE student
ADD COLUMN gpa real;
```

Integrity Constraints (ICs)

- IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.*
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!

Primary Key Constraints Revisited

- A set of fields is a *key* for a relation if :
  1. No two distinct tuples can have same values in all key fields, and
  2. no subset of the key is also a key.
  - Otherwise, it's a *superkey*.
  - If there is >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.
- e.g., *ID* is a key for student. (What about *name*?) The set {*ID*, *name*} is a superkey.

Primary and Candidate Keys in SQL

- Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.
- What's the difference between the two statements below?

```
CREATE TABLE Enrolled
  (ID CHAR(5),
   course_id CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (ID, course_id) );
```

```
CREATE TABLE Enrolled
  (ID CHAR(5),
   course_id CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (ID),
   UNIQUE (course_id, grade));
```

Used carelessly, an IC can prevent the storage of database instances that arise in practice!

Foreign Keys, Referential Integrity

- *Foreign key* : Set of fields in one relation that is used to "refer" to a tuple in another relation. (Usually correspond to primary key of the second relation.) Like a `logical pointer' .
- e.g. *s_ID* is a foreign key referring to student(ID):
  - *advisor*(*i_ID*: string, *s_ID:* string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.
  - Can you name a data model w/o referential integrity?

Foreign Keys in SQL

```
CREATE TABLE department
    (dept_name VARCHAR2(20),
     building VARCHAR2(20),
     budget integer,
     PRIMARY KEY(dept_name));

CREATE TABLE instructor
    (ID CHAR(5),
    name VARCHAR2(50),
    salary integer,
    dept_name VARCHAR2(20),
    PRIMARY KEY(ID),
    FOREIGN KEY(dept_name) REFERENCES department);

CREATE TABLE advisor
    (s_ID CHAR(5),
     i_ID CHAR(5),
     PRIMARY KEY(s_ID),
     FOREIGN KEY(i_ID) REFERENCES instructor(ID),
     FOREIGN KEY(s_ID) REFERENCES student(ID));
```

Enforcing Referential Integrity

- Consider *instructor* and *advisor*; *i_ID* in *advisor* is a foreign key that references *instructor*.
- What should be done if an *advisor* tuple with a non-existent instructor id is inserted? (*Reject it!*)
- What should be done if an *instructor* tuple is deleted?
  - Also delete the *advisor* tuples that refer to it.
  - Disallow deletion of an instructor tuple that is referred to.
  - Set i_ID in *advisor* tuples that refer to it to a default i_ID.
  - Set i_ID in *advisor* tuples that refer to it to a special value *null*, denoting *`unknown'* or *`inapplicable'*.)
- Similar if primary key of instructor tuple is updated.

Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and 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)

```
CREATE TABLE advisor
   (s_ID CHAR(5),
    i_ID CHAR(5),
    PRIMARY KEY(s_ID),
    FOREIGN KEY(i_ID)
          REFERENCES instructor(ID)
          ON DELETE SET NULL,
    FOREIGN KEY(s_ID)
          REFERENCES student(ID)
          ON DELETE CASCADE);
```

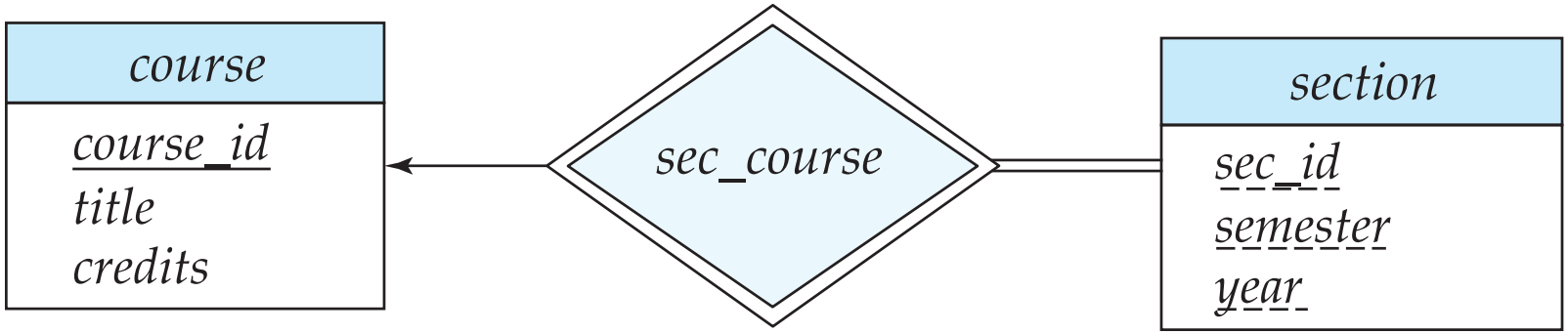
Oracle does not support this.
*Oracle also does not support ON UPDATE

Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about *all possible* instances!
  - From example, we know *name* is not a key, but the assertion that *sid* is a key is given to us.
- Key and foreign key ICs are the most common; more general ICs supported too.

Review: Weak Entities

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.



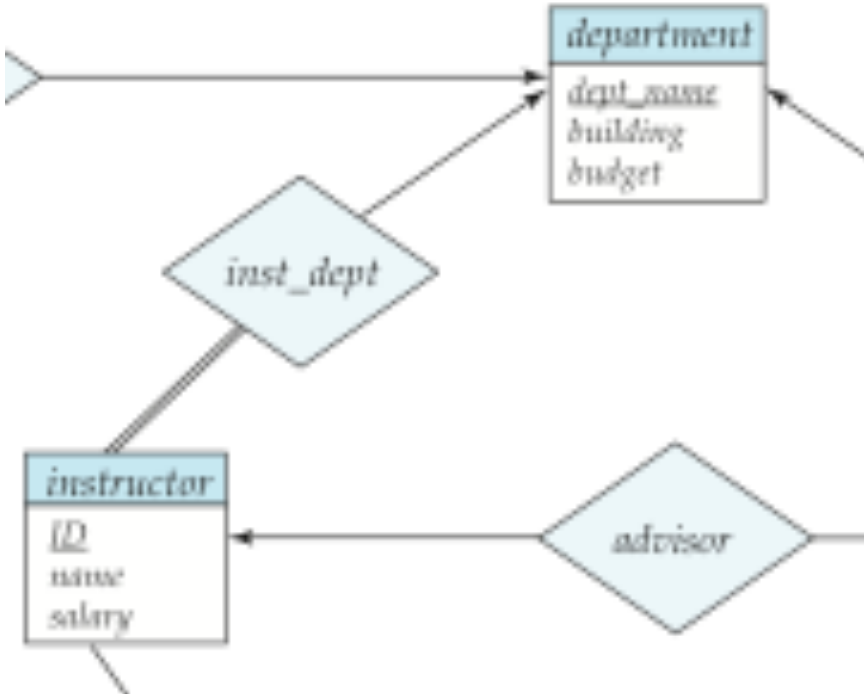
Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted.

course_sec (*course_id* : string, *sec_id*: string, *semester*: string, *year*: integer)

```
CREATE TABLE course_sec (
  course_id CHAR(10),
  sec_id CHAR(5),
  semester CHAR(6),
  year integer,
  PRIMARY KEY  (course_id, sec_id, semester, year),
  FOREIGN KEY  (course_id) REFERENCES course
    ON DELETE CASCADE);
```

Total Participation




```
CREATE TABLE instructor
    (ID CHAR(5),
    name VARCHAR2(50),
    dept_name VARCHAR(20) NOT NULL,
    salary integer,
    PRIMARY KEY(ID),
    FOREIGN KEY(dept_name) REFERENCES department);
```

Translating ISA Hierarchies to Relations

- General approach:
  - 3 relations: Employees, Hourly_Emps and Contract_Emps.
    - *Hourly_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages*, *hours_worked*, *ssn)*; must delete Hourly_Emps tuple if referenced Employees tuple is deleted.
    - Contract_Emps: Every employee is recorded in Employees. Extra info recorded in Contract_Emps (contract_id); must delete Contract_Emps tuple if referenced Employees tuple is deleted.
    - Queries involving all employees easy, those involving just Hourly_Emps require a join to get some attributes.
- Alternative: Just Hourly_Emps and Contract_Emps.
  - *Hourly_Emps*: *ssn, name, lot, hourly_wages, hours_worked.*
  - Similar for Contract_Emps
  - Each employee must be in one of these two subclasses otherwise need to create another schema just to store the basic employee info.

Views

- A *view* is just a relation, but we store a *definition*, rather than a set of tuples.

```
CREATE VIEW YoungStudents (name, grade)
        AS SELECT S.name, E.grade
        FROM Students S, Enrolled E
        WHERE S.sid = E.sid and S.age<21;
```

- Views can be dropped using the DROP VIEW command.

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given YoungStudents, but not Students or Enrolled, we can find young students who are enrolled, but not the *cids* of the courses they are enrolled in.

Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used.
- Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
  - Two important ICs: primary and foreign keys
  - In addition, we *always* have domain constraints.
- Powerful and natural query languages exist.
- Rules to translate ER to relational model