ECE30030/ITP30010 Database Systems

More SQL & Designing a DB

Reading: Chapters 3, 6

Charmgil Hong

charmgil@handong.edu

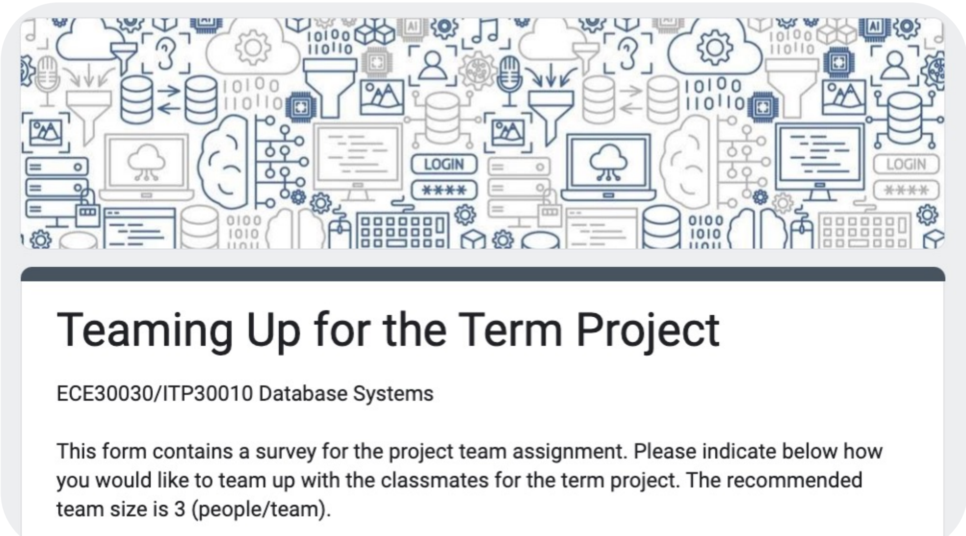
Spring, 2025

Handong Global University

Announcements

- HW#2 is due this Thursday (April 10)
  - HW#3 is pre-released (official release: April 10; due: April 24)

- Make teams for the term project
  - https://forms.gle/T742G8LQBikzfrUv9 Reponse due: Thursday, April 17
  - Problem & data release: Week #8 (tentative)



Declaring Keys

- An attribute or list of attributes may be declared as PRIMARY KEY or UNIQUE
  - Meaning: no two tuples of the relation may agree in all the attribute(s) on the list
    - That is, the attribute(s) do(es) not allow duplicates in values
    - PRIMARY KEY/UNIQUE can be used as an identifier for each row
  - Comparison: PRIMARY KEY vs UNIQUE

PRIMARY KEY	UNIQUE
Used to serve as a unique identifier for each row in a relation	Uniquely determines a row which is not primary key
Cannot accept NULL	Can accept NULL values (some DBMSs accept only one NULL value)
A relation can have only one primary key	A relation can have more than one unique attributes
Clustered index	Non-clustered index

Integrity Constraints

- **NOT NULL** – disallowing null values
  - Null values indicate that the data is not known
  - These can cause problems in querying database
  - The Primary Key columns automatically prevent null being entered
  - *C.f.*, **NULL** – can be used to explicitly allow null values

```
CREATE TABLE studio (
    ID       NUMERIC(5,0) PRIMARY KEY,
    name     VARCHAR(20) NOT NULL,
    city     VARCHAR(20) NULL,
    state    CHAR(2) NOT NULL
);
```

Integrity Constraints

- **DEFAULT** – A default value can be inserted in any column with this keyword
  - *E.g.,* **CREATE TABLE** *movies*(

```
            movie_title      VARCHAR(40) NOT NULL,
            release_date     DATE  DEFAULT sysdate  NULL,
            genre            VARCHAR(20) DEFAULT 'Comedy'
                             CHECK genre IN ('Comedy', 'Action', 'Drama')
      )
```

  - In MySQL,
    - **CREATE TABLE** *movies*(

```
            movie_title      VARCHAR(40) NOT NULL,
            release_date     DATE  DEFAULT CURRENT_TIMESTAMP  NULL,
            genre            VARCHAR(20) DEFAULT 'Comedy'
                             CHECK genre IN ('Comedy', 'Action', 'Drama')
      )
```

Integrity Constraints

- **CHECK** – Allows the inserted value to be checked
  - *E.g.*, **CREATE TABLE** *movies*(

    movie_title **VARCHAR**(40) **PRIMARY KEY**,
    release_date **DATE**,
    budget **INTEGER CHECK** (*budget* > 50000)

)

  - Table-level constraints can be defined; *E.g.*,
    - **CREATE TABLE** *movies*(

      movie_title **VARCHAR**(40) **PRIMARY KEY**,
      release_date **DATE**,
      budget **INTEGER CHECK** (*budget* > 50000),
      CONSTRAINT release_date_const
      CHECK (release_date **BETWEEN** '01-Jan-2000' **AND** '31-Dec-2009')

)

Declaring Keys

- **CREATE TABLE** *student* (
  ID **VARCHAR**(5),
  name **VARCHAR**(20) **NOT NULL,**
  dept_name **VARCHAR**(20),
  tot_cred **NUMERIC**(3,0),
  PRIMARY KEY *(ID),*
  FOREIGN KEY *(dept_name*) **REFERENCES** *department*);

- **CREATE TABLE** *student* (
  ID **VARCHAR**(5) **PRIMARY KEY,**
  name **VARCHAR**(20) **NOT NULL,**
  dept_name **VARCHAR**(20),
  tot_cred **NUMERIC**(3,0),
  FOREIGN KEY *(dept_name*) **REFERENCES** *department*);

More Examples

- **CREATE TABLE** *takes* (

```
        ID              VARCHAR(5),
        course_id       VARCHAR(8),
        sec_id          VARCHAR(8),
        semester        VARCHAR(6),
        year            NUMERIC(4,0),
        grade           VARCHAR(2),
        PRIMARY KEY (ID, course_id, sec_id, semester, year),
        FOREIGN KEY (ID) REFERENCES  student,
        FOREIGN KEY (course_id, sec_id, semester, year)
                        REFERENCES section);
```

More Examples

- **CREATE TABLE** *course* (
  course_id **VARCHAR**(8),
  title **VARCHAR**(50),
  dept_name **VARCHAR**(20) **DEFAULT** ‘Comp. Sci’,
  credits **NUMERIC**(2,0),
  PRIMARY KEY *(course_id),*
  FOREIGN KEY *(dept_name*) **REFERENCES** *department*);

More Examples

- **CREATE TABLE** *neighbors*(
  name **CHAR**(30) **PRIMARY KEY**,
  addr **CHAR**(50) **DEFAULT** ‘123 Sesame St.’,
  phone **CHAR**(16));

  - Inserting Elmo is a neighbor:
    - INSERT INTO neighbors (name)
      VALUES (‘Elmo’);

name	addr	phone
‘Elmo’	‘123 Sesame St.’	NULL

More Examples

- **CREATE TABLE** *neighbors*(
  name **CHAR**(30) **PRIMARY KEY**,
  addr **CHAR**(50) **DEFAULT** ‘123 Sesame St.’,
  phone **CHAR**(16) **NOT NULL**);

  - Inserting Elmo is a neighbor:
    - INSERT INTO neighbors (name)
      VALUES (‘Elmo’);

      ➔ If phone were NOT NULL, this insertion would have been rejected

Column-level vs. Table-level Foreign Key Declarations

- Column-level declaration
  - The declaration clause is written directly next to the column definition
    - *E.g.*, `CREATE TABLE student (`

      ```
            dept_name VARCHAR(20) REFERENCES department(dept_name));
      ```

    - Simple and intuitive syntax
    - Clearly ties the declared property to the individual column
    - Cannot define composite keys

Column-level vs. Table-level Foreign Key Declarations

- Table-level declaration
  - The declaration is at the bottom of the table definition, outside individual column lines
    - *E.g.,*
```
CREATE TABLE student (
        dept_name VARCHAR(20),
        FOREIGN KEY (dept_name) REFERENCES department(dept_name));
```
    - *E.g.,*
```
CREATE TABLE enrollment (
        student_id VARCHAR(10),
        course_id VARCHAR(10),
        FOREIGN KEY (student_id, course_id)
          REFERENCES takes(student_id, course_id));
```
    - Required for composite keys (more than one column)
    - Allows naming constraints
      - *E.g.,*
```
CONSTRAINT fk_dept
        FOREIGN KEY (dept_name) REFERENCES department(dept_name)
```

Agenda

- Nested subqueries
- Set membership (SOME, ALL, EXISTS)
- Designing a database
- E-R diagrams

Running Examples

- Relations (tables): *instructor, teaches*

Instructor relation

ID	name	dept_name	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

teaches relation

ID	course_id	sec_id	semester	year
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
10101	CS-101	1	Fall	2017
45565	CS-101	1	Spring	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
10101	CS-315	1	Spring	2018
45565	CS-319	1	Spring	2018
83821	CS-319	2	Spring	2018
10101	CS-347	1	Fall	2017
98345	EE-181	1	Spring	2017
12121	FIN-201	1	Spring	2018
32343	HIS-351	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017

Running Examples

- Relations (tables): *course, takes*

course relation

course_id	title	dept_name	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

takes relation

ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	<null>

Running Examples

- Relations (tables): *student*

student relation

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries. A subquery is a SELECT-FROM-WHERE expression that is nested within another query
- The nesting can be done in the following SQL query

  SELECT $A_1, A_2, \ldots, A_n$
  FROM $r_1, r_2, \ldots, r_m$
  WHERE P

  as follows:
  - **FROM clause:** r_i can be replaced by any valid subquery
  - **WHERE clause:** P can be replaced with an expression of the form:
    B <operation> (subquery)
    B is an attribute and <operation> is to be defined later
  - **SELECT clause:**
    A_i can be replaced by a subquery that generates a single value (scalar subquery)

Subqueries in the FROM Clause

- *E.g.*, Find the average instructors' salaries of those departments where the average salary is greater than $42,000
  - **SELECT** *D.dept_name*, *D.avg_salary*
    FROM (**SELECT** *dept_name*, **AVG**(*salary*) **AS** *avg_salary*
    FROM *instructor*
    GROUP BY *dept_name*) **AS** *D*
    WHERE *D.avg_salary* > 42000;

dept_name	avg_salary
Biology	72000.000000
Comp. Sci.	77333.333333
Elec. Eng.	80000.000000
Finance	85000.000000
History	61000.000000
Physics	91000.000000

WITH Clause

- The **WITH** clause provides a way of defining a temporary relation
  - The relation is available only to the query in which the **WITH** clause occurs

- *E.g.*, Find all departments with the maximum budget
  - **WITH** *max_budget* (*value*) **AS**
    (**SELECT MAX**(*budget*)
    FROM *department*)
    SELECT *department.dept_name*
    FROM *department*, *max_budget*
    WHERE *department.budget* = *max_budget.value*;

dept_name
Finance

WITH Clause

- *E.g.*, Find the average instructors' salaries of those departments where the average salary is greater than $42,000
  - **WITH** *D*(*dept_name, avg_salary*) **AS**
    (**SELECT** *dept_name*, **AVG**(*salary*)
    FROM *instructor*
    GROUP BY *dept_name*)
    SELECT *dept_name, avg_salary*
    FROM *D*
    WHERE *avg_salary* > 42000;

dept_name	avg_salary
Biology	72000.000000
Comp. Sci.	77333.333333
Elec. Eng.	80000.000000
Finance	85000.000000
History	61000.000000
Physics	91000.000000

Scalar Subquery

- Scalar subquery is used where a single value is expected
  - Runtime error occurs if a subquery returns more than one result tuple
- *E.g.*, List all departments along with the number of instructors in each department
  - **SELECT** *dept_name*,
    (**SELECT COUNT**(*)
    FROM *instructor*
    WHERE *department.dept_name* = *instructor.dept_name*)
    AS *num_instructors*
    FROM *department*;

dept_name	num_instructors
Biology	1
Comp. Sci.	3
Elec. Eng.	1
Finance	2
History	2
Music	1
Physics	2

Agenda

- Nested subqueries
- **Set membership (SOME, ALL, EXISTS)**
- Designing a database
- E-R diagrams

Sets and Relations

- Set theory
  - A branch of mathematics that studies sets (their relationships, and operations)

- The relational database model is based on set theory
  - Data is organized into tables; each table can be considered a set of rows
  - Fundamental set theory operations (UNION, INTERSET, EXCEPT) are directly implemented in SQL
  - The WHERE and HAVING clauses in SQL are analogous to the selection of certain elements based on conditions in set theory
  - The JOIN operation is based on the Cartesian product in set theory where more specific relationships can be defined using predicates

Set Membership

- Find courses offered in Fall 2017 and in Spring 2018
  - **SELECT DISTINCT** *course_id*
    FROM *teaches*
    WHERE *semester* = 'Fall' **AND** *year*= 2017 **AND**
    course_id **IN** (**SELECT** *course_id*
    FROM *teaches*
    WHERE *semester* = 'Spring' **AND** *year*= 2018);

course_id
CS-101

Set Membership

- Find courses offered in Fall 2017 but not in Spring 2018
  - **SELECT DISTINCT** *course_id*
FROM *teaches*
WHERE *semester* = 'Fall' **AND** *year*= 2017 **AND**
course_id ***NOT IN*** (**SELECT** *course_id*
FROM *teaches*
WHERE *semester* = 'Spring' **AND** *year*= 2018);

course_id
CS-347
PHY-101

Set Membership

- Name all instructors whose name is neither “Mozart” nor Einstein”
    - **SELECT DISTINCT** *name*
      FROM *instructor*
      WHERE *name* **NOT IN** ('Mozart', 'Einstein’);

name
Srinivasan
Wu
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

Set Membership

- Find the total number of unique students who have taken course sections taught by the instructor with *ID* 10101
  - **SELECT COUNT**(**DISTINCT** *ID*)
    FROM *takes*
    WHERE (*course_id*, *sec_id*, *semester*, *year*) **IN**
    (**SELECT** *course_id*, *sec_id*, *semester*, *year*
    FROM *teaches*
    WHERE *teaches*.*ID*= 10101);

`COUNT(DISTINCT ID)`
6

  - Note: *Above query can be written in a much simpler manner*
    The formulation above is simply to illustrate SQL features

Set Comparison – SOME

- Find names of instructors with salary greater than that of SOME (at least one) instructor in the Biology department
  - **SELECT DISTINCT** *T.name*
    FROM *instructor* **AS** *T*, *instructor* **AS** *S*
    WHERE *T.salary* > *S.salary* **AND** *S.dept name* = 'Biology';

- Same query using > **SOME** clause
  - **SELECT** *name*
    FROM *instructor*
    WHERE *salary* > **SOME** (**SELECT** *salary*
    FROM *instructor*
    WHERE *dept_name* = 'Biology');

name
Wu
Einstein
Gold
Katz
Singh
Brandt
Kim

Interpretation of SOME

- F <comp> **SOME** $r \Leftrightarrow \exists t \in r$ such that (F <comp> t)
  Where <comp> can be: $<, \leq, >, =, \neq$

$$(5 < \textbf{SOME} \begin{array}{|c|}\hline 0\\\hline 5\\\hline 6\\\hline\end{array}) = \text{true}$$

(read: 5 < some tuple in the relation)

$$(5 < \textbf{SOME} \begin{array}{|c|}\hline 0\\\hline 5\\\hline\end{array}) = \text{false}$$

$$(5 = \textbf{SOME} \begin{array}{|c|}\hline 0\\\hline 5\\\hline\end{array}) = \text{true}$$

$$(5 \neq \textbf{SOME} \begin{array}{|c|}\hline 0\\\hline 5\\\hline\end{array}) = \text{true (since } 0 \neq 5)$$

(= **SOME**) $\equiv$ **IN**
However, ($\neq$ **SOME**) $\not\equiv$ **NOT IN**

Set Comparison – ALL

- Find the names of ALL instructors whose salary is greater than the salary of ALL instructors in the Biology department
  - **SELECT** *name*
    FROM *instructor*
    WHERE *salary* > **ALL** (**SELECT** *salary*
    FROM *instructor*
    WHERE *dept name* = 'Biology');

name
Wu
Einstein
Gold
Katz
Singh
Brandt
Kim

Interpretation of ALL

- F <comp> **ALL** $r \Leftrightarrow \forall t \in r$ (F <comp> t)

$$(5 < \textbf{ALL} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline\end{array}) = \text{false}$$

$$(5 < \textbf{ALL} \begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline\end{array}) = \text{true}$$

$$(5 = \textbf{ALL} \begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline\end{array}) = \text{false}$$

$$(5 \neq \textbf{ALL} \begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline\end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$$

($\neq$ **ALL**) $\equiv$ **NOT IN**
However, (= **ALL**) $\not\equiv$ **IN**

Test for Empty Relations

- The **EXISTS** construct returns the value *true* if the argument subquery is nonempty
  - **EXISTS** $r \Leftrightarrow r \neq \emptyset$
  - **NOT EXISTS** $r \Leftrightarrow r = \emptyset$

Use of EXISTS

- Yet another way of specifying the query "Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester"
  - **SELECT** *course_id*
FROM *teaches* **AS** *S*
WHERE *semester* = 'Fall' **AND** *year* = 2017 **AND**
        EXISTS (**SELECT** *
                FROM *teaches* **AS** *T*
                WHERE *semester* = 'Spring' **AND** *year*= 2018
                        AND *S.course_id* = *T.course_id*);

course_id
CS-101

Use of NOT EXISTS

- Find all students who have taken all courses offered in the Music department
  - ```
    SELECT DISTINCT S.ID, S.name
    FROM student AS S
    WHERE NOT EXISTS ( SELECT course_id
                       FROM course
                       WHERE dept_name = 'Music'
                             AND course_id NOT IN
                                  (SELECT T.course_id
                                   FROM takes AS T
                                   WHERE S.ID = T.ID));
    ```

ID	name
55739	Sanchez
    ```

Use of NOT EXISTS

- Note: Renaming (**AS**) is optional in certain contexts
  - **SELECT DISTINCT** *ID*, *name*
    FROM *student*
    WHERE NOT EXISTS (**SELECT** *course_id*
    FROM *course*
    WHERE *dept_name* = ‘Music'
    AND *course_id* **NOT IN**
    (**SELECT** *course_id*
    FROM *takes*
    WHERE *student*.*ID* = *takes*.*ID*));

  - Exception: the following query results in an empty relation
    - **SELECT DISTINCT** *name*
      FROM *instructor*
      WHERE *salary* > *salary* **AND** *dept_name* = 'Biology';

Use of NOT EXISTS

- Some systems support the **EXCEPT** clause (MySQL does not)
- Find all students who have taken all courses offered in the Music department
  - **SELECT DISTINCT** *S.ID*, *S.name*
    FROM *student* **AS** *S*
    WHERE NOT EXISTS ((**SELECT** *course_id*
    FROM *course*
    WHERE *dept_name* = 'Music')
    EXCEPT
    (**SELECT** *T.course_id*
    FROM *takes* **AS** *T*
    WHERE *S.ID* = *T.ID*));

Test for Absence of Duplicate Tuples

- The **UNIQUE** construct tests whether a subquery has any duplicate tuples in its result
  - **UNIQUE** evaluates to "true" if a given subquery contains no duplicates
  - MySQL does not support the UNIQUE test (UNIQUE in MySQL is a constraint specifier)

- Find all courses that were offered at most once in 2017
  - **SELECT** *T.course_id*
    FROM *course* **AS** *T*
    WHERE **UNIQUE** (**SELECT** *R.course_id*
    FROM *teaches* **AS** *R*
    WHERE *T.course_id*= *R.course_id* **AND** *R.year* = 2017);

Agenda

- Nested subqueries
- Set membership (SOME, ALL, EXISTS)
- **Designing a database**
- E-R diagrams

Design Phases

- Initial phase: characterize fully the data needs of the prospective database users
- Second phase: choose a data model
  - Apply the concepts of the chosen data model
  - Translate the requirements into a conceptual schema of the database
  - A fully developed conceptual schema indicates the functional requirements of the enterprise
    - Describe the kinds of operations (or transactions) that will be performed on the data

Problem analysis
- Characterize the data needs

→

Choosing a data model
- Translate the requirements into a conceptual schema

→

Design
- Logical design
- Physical design

Design Phases

- Final Phase: Move from an abstract data model to the implementation of the database
  - Logical Design – Deciding on the database schema
    - Database design requires that we find a "good" collection of relation schemas
    - *Business decision – What attributes should we record in the database?*
    - *Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?*
  - Physical Design – Deciding on the physical layout of the database

Problem analysis

- Characterize the data needs

Choosing a data model

- Translate the requirements into a conceptual schema

Design

- Logical design
- Physical design

Design Phases

- In designing a database schema, we must ensure that we avoid two major pitfalls:
  - Redundancy: a bad design may result in repeated information
    - Redundant representation of information may lead to data inconsistency among the various copies of information
  - Incompleteness: a bad design may make certain aspects of the enterprise difficult or impossible to model

- Avoiding bad designs is not enough. There may be a large number of good designs from which we must choose

Design Approaches

- Entity Relationship Model
  - Models an enterprise as a collection of *entities* and *relationships*
    - Entity: a “thing” or “object” in the enterprise that is distinguishable from other objects
      - Described by a set of *attributes*
    - Relationship: an association among several entities
  - Represented diagrammatically by an *entity-relationship diagram* (E-R diagram)

- Normalization Theory
  - Formalize what designs are bad, and test for them