Functions, Integrity

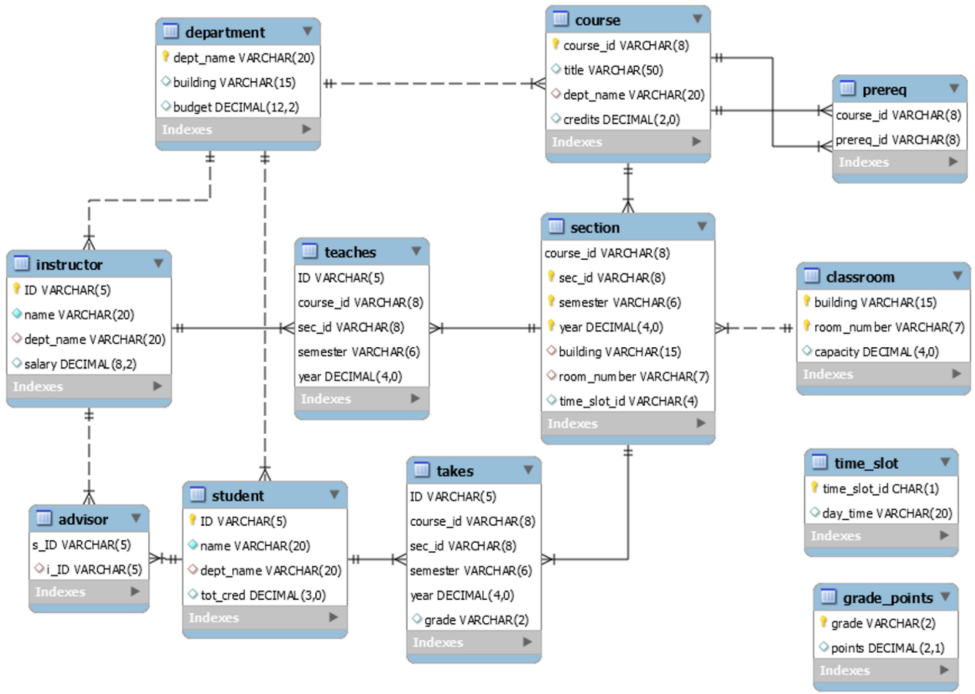
CST 363

Topics

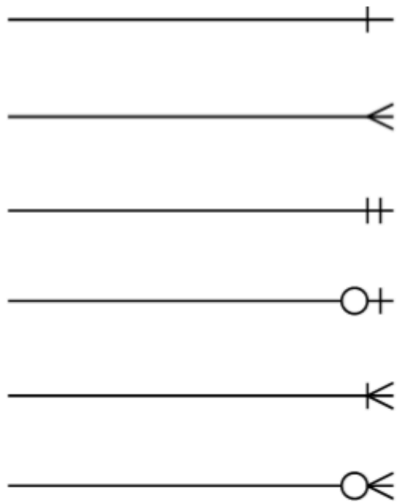
- `SELECT DISTINCT`
- *Scalar* functions
- `LIKE`, `BETWEEN`, `TUPLE` predicates
- `NULL` values
- Constraints
  - `CHECK` constraints
  - `UNIQUE`
  - Foreign keys
- Cascading actions

Brief Entity Relationship (ER) Diagram Aside

Example ER Diagram for University Database



ER Diagram – Crow's Foot Notation



one

many

one (and only one)

zero or one

one or many

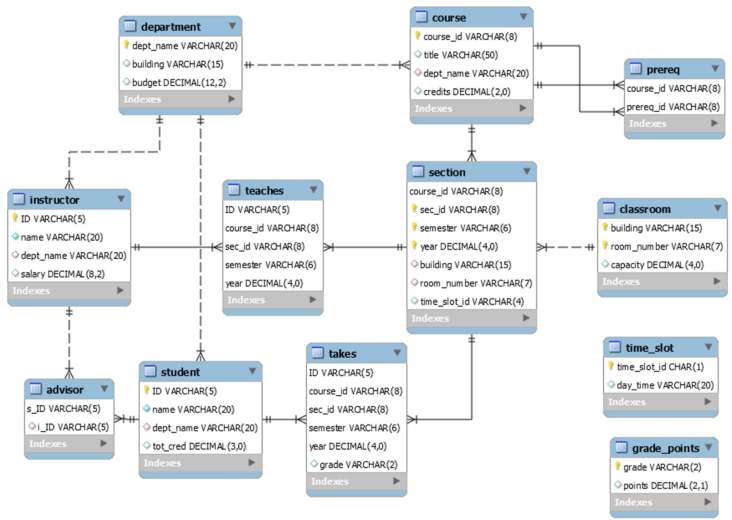
zero or many

ER Diagram – Weak vs Strong Relationships

- Dotted line – **weak / non-identifying** relationship
  - The *child* table references the *parent's* primary key as a foreign key,
    but the child's own primary key does not depend on it. The child still has its own identity.
  - FK exists, but not part of PK

- Solid line – strong / identifying relationship
  - FK is part of PK

Example ER Diagram for University Database (Again)



- Identifying (solid)
  - `course → section`
    - A section is identified by `(course_id, sec_id, semester, year)` (parent key participates).
    - In your diagram, section includes `course_id` as part of its **PK** → identifying.
- Non-identifying (dotted)
  - `department → course`
    - course has its own **PK** (`course_id`).
    - `dept_name` is **FK** but not part of course **PK** → non-identifying.

SELECT DISTINCT

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword `DISTINCT` after `SELECT`.
  - `DISTINCT` applies to the entire selected row (all selected columns), not each column independently.
- Find the department names of all instructors, and remove duplicates

```
SELECT dept_name
FROM instructor;
```

instructor

dept_name
Comp. Sci.
Finance
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

```
SELECT DISTINCT dept_name
FROM instructor;
```

instructor

dept_name
Comp. Sci.
Finance
Physics
History
Biology
Elec. Eng.

"Scalar" Functions

A scalar function returns one value per row (or per call) – not a table/result set.

- SQL supports a variety of string operations such as
  - concatenation
    - `CONCAT(last_name, ', ', first_name)`
  - converting from upper to lower case (and vice versa)
    - `UPPER(name), LOWER(name)`
  - finding string length, extracting substrings, find a substring etc.
    - `LENGTH(name), SUBSTRING(name, 3, 4)`
    - `SELECT POSITION('SU' IN 'CSUMB');` → returns 2
    - SQL starts indexing with 1, not 0

Scalar Functions (cont.)

- Numeric functions include
  - `ROUND(23.66666, 1)` → 23.7
  - `ROUND(23.66666)` → 24

LIKE Predicate

- The predicate `LIKE` uses patterns to match strings that are described using two special characters:
  - percent (`%`). The % character matches any substring.
  - underscore (`_`). The _ character matches any character.
- Find the names of all instructors whose name includes the substring "dar".

```
SELECT instructor_name
FROM instructor
WHERE instructor_name LIKE '%dar%';
```

LIKE Predicate – Examples

- `LIKE '100\%' ESCAPE '\'` matches the string 100%
- `LIKE 'Intro%'` matches any string beginning with “Intro”.
- `LIKE '%Comp%'` matches any string containing “Comp” as a substring.
- `LIKE '___'` matches any string of exactly three characters.
- `LIKE '___%'` matches any string of at least three characters

BETWEEN Predicate

- SQL includes a `BETWEEN` comparison operator

- Example: Find the names of all instructors with salary between $90,000 and $100,000

```
SELECT instructor_name
FROM instructor
WHERE salary BETWEEN 90000 AND 100000;
```

equivalent to:

```
SELECT instructor_name
FROM instructor
WHERE salary ⩾ 90000  AND salary ⩽ 100000;
```

Tuple comparison

```sql
SELECT instructor_name, dept_name
FROM instructor
WHERE (ID, dept_name) = ('45565', 'Comp. Sci.');
```

equivalent to:

```sql
SELECT instructor_name, dept_name
FROM instructor
WHERE ID = '45565' AND dept_name = 'Comp. Sci.';
```

Null Values

- The predicate `IS NULL` can be used to check for null values.

- Example: Find all instructors whose salary is `NULL` .

```
SELECT instructor_name
FROM instructor
WHERE salary IS NULL;
```

- The predicate `IS NOT NULL` succeeds if the value on which it is applied is not null.
- The result of an expression involving `NULL` is `NULL`
- *Example:* `5 + NULL` returns `NULL`

Null Values (Cont.)

- SQL treats as unknown the result of any comparison involving a null value (other than predicates `IS NULL` and `IS NOT NULL`).
  - Example: `5 < NULL` or `NULL <> NULL` or `NULL = NULL`
- The predicate in a `WHERE` clause can involve Boolean operations (`AND`, `OR`); thus the definitions of the Boolean operations need to be extended to deal with the value **unknown**.
  - `AND` : (true AND unknown) = unknown,
    (false AND unknown) = false, (unknown AND unknown) = unknown
  - `OR` : (unknown OR true) = true, (unknown OR false) = unknown, (unknown OR unknown) = unknown
- Result of `WHERE` clause predicate is treated as false if it evaluates to unknown
- **Note:** `= NULL` never works; use `IS NULL`.

Integrity Constraints

- An integrity constraint is a condition on data.
- If an integrity constraint doesn't hold, there is a problem (or inconsistency) with the data.
- Integrity constraint can involve
  - invalid key (primary key or foreign key) value
  - invalid data value (negative salary value for example)
  - uniqueness (2 different students same social security number)

You define the integrity constraints; the database enforces them.

Recall: Constraints

- `PRIMARY KEY` = `UNIQUE` + `NOT NULL`
- `FOREIGN KEY` = must reference an existing parent key
- `CHECK` = row-level predicate must be true

SQL CHECK constraints (Example 1 of 3)

```sql
CREATE TABLE department (
   dept_name VARCHAR(20),
   building VARCHAR(15),
   budget NUMERIC(12, 2) CHECK (budget > 0),
   PRIMARY KEY (dept_name)
);
```

- A department's budget must be greater than 0

SQL CHECK constraints (Example 2 of 3)

```
CREATE TABLE section (
   course_id VARCHAR(8),
   sec_id VARCHAR(8),
   semester VARCHAR(6) CHECK (
       semester IN ('Fall', 'Winter', 'Spring', 'Summer')
   ),
   ...
);
```

- Semester must be one of Fall/Winter/Spring/Summer.

SQL CHECK constraints (Example 3 of 3)

```
CREATE TABLE course (
   course_id VARCHAR(8),
   title VARCHAR(50),
   dept_name VARCHAR(20),
   credits NUMERIC(2, 0),
   CHECK (
       dept_name <> 'Comp. Sci.' OR
       course_id LIKE 'CS-%'
   )
);
```

- If dept_name is "Comp. Sci." then course_id must start with "CS-"
- `<>` can also be written in PostgreSQL as `≠`

SQL UNIQUE constraints

```
CREATE TABLE department (
   dept_name VARCHAR(20) PRIMARY KEY,
   dept_abbrv VARCHAR(4),
   building VARCHAR(15),
   budget NUMERIC(12, 2),
   UNIQUE (dept_abbrv)
);
```

No two rows can have the same non-null `dept_abbr` values

Things to remember:

- `UNIQUE` constraints look like primary key constraints except unique columns can be a null value, primary key's do not allow null values.
- A table can have many `UNIQUE` constraints
- `NULL = x` is always false, so `UNIQUE` only cares about non-null values

SQL FOREIGN KEY constraints

A foreign key column(s), if not null, must match a candidate key (typically the PK) of the referenced table.

```
CREATE TABLE course (
    ...,
   dept_name VARCHAR(20),
   FOREIGN KEY (dept_name) REFERENCES department (dept_name),
    ...
);
```

- `INSERT INTO course VALUES ( ... , NULL, ... );` → OK
- `INSERT INTO course VALUES ( ... , 'Comp. Sci.', ... );` → OK
- `INSERT INTO course VALUES ( ... , 'CompSci.', ... );` → Constraint violation, misspelled

Cascading Actions in Referential Integrity

- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.
- An alternative, in case of `DELETE` or `UPDATE` is to `CASCADE`.
  - If the department is deleted, delete all courses related to that department.
  - If the department name is updated in the department table, change the foreign key values in the course's table.

```
CREATE TABLE course (
    ... ,
    dept_name VARCHAR(20),
    FOREIGN KEY (dept_name) REFERENCES department(dept_name)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    ...
);
```

Cascading Actions in Referential Integrity (cont.)

- Instead of `CASCADE`, we can use `SET NULL` or `SET DEFAULT`:
  - `SET NULL`: If the referenced row is deleted or updated, the foreign key is set to `NULL`.
  - `SET DEFAULT`: If the referenced row is deleted or updated, the foreign key is set to its default value (if a default is defined).