DDL & DML

⏪ Recap

- Subquery
- Aggregation
- Grouping
- Joining

Aggregation with `GROUP BY` and `HAVING`

> I know that she attended the SQL Symphony Concert 3 times in December 2017.

```
select person_id
from facebook_event_checkin
where event_name like "%SQL Symphony Concert%"
and date between 20170101 and 20171231
group by person_id
having count(*)=3
```

Subquery

```
select *
from person
where license_id in (...)
and id in (...)
```

Join

```
select *
from person join drivers_license
on person.license_id = drivers_license.id
where drivers_license.hair_color = 'red';
```

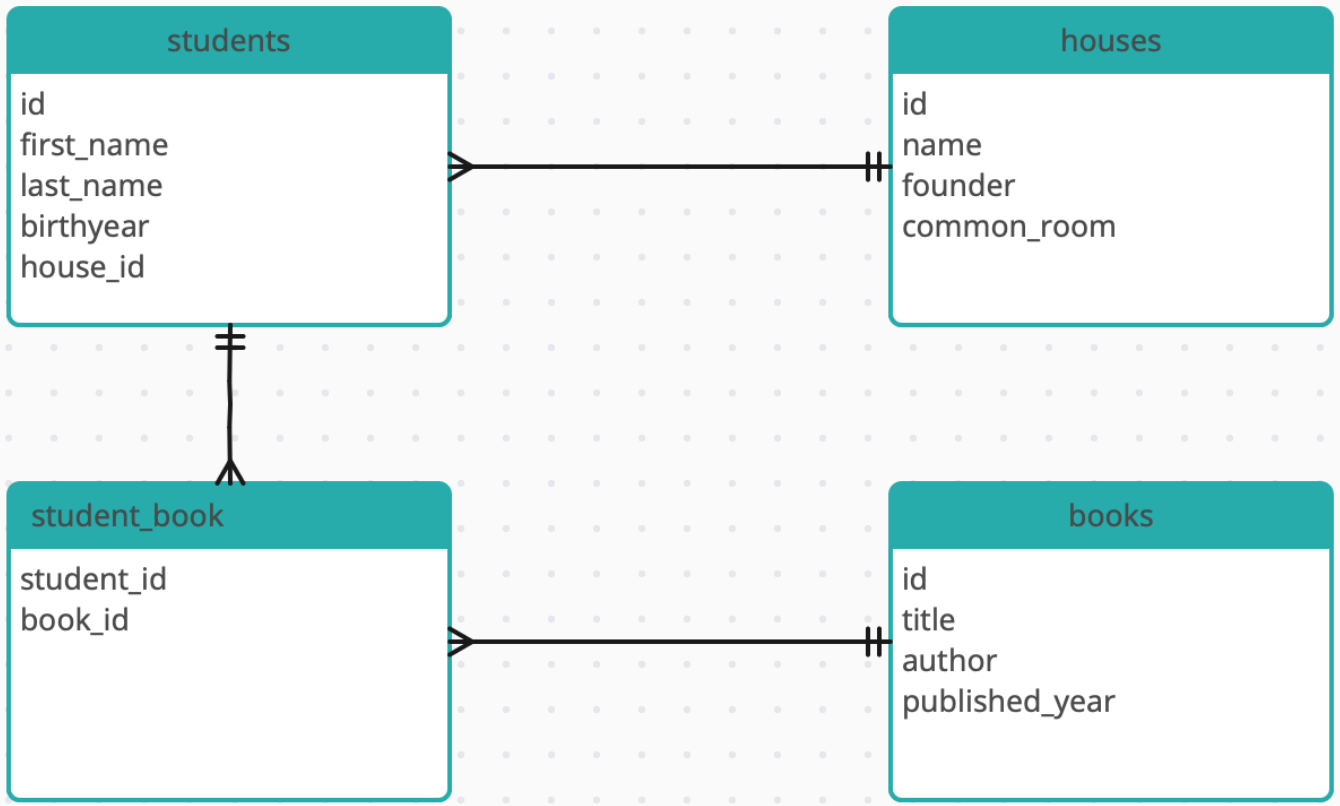
DDL: Data Definition Language

- `CREATE TABLE` : **create a new table**
- `ALTER TABLE` : modify a table
- `DROP TABLE` : remove a table

DML: Data Manipulation Language

- `INSERT INTO` : **insert data into a table**
- `UPDATE` : update data in a table
- `DELETE FROM` : remove data from a table

Create books and student_book tables



`CREATE TABLE`: Create a new table

```
CREATE TABLE table_name (
    column_name1 data_type,
    column_name2 data_type,
    ...
    column_nameN data_type
);
```

```
CREATE TABLE books (
    id numeric,
    title text,
    author text,
    published_year numeric
);
```

Define data types for columns

`numeric`

- `int`
- `float`

...

`text`

- `varchar(n)`
- `text`

...

---

https://www.sqlite.org/datatype3.html

Create table with constraints

```
CREATE TABLE books (
    id numeric,
    title text,
    author text,
    published_year numeric,
    -- table-level constraints defined at the end
    PRIMARY KEY (id),                  -- id is the primary key
    CHECK (published_year >= 1900)  -- published_year must be >= 1900
);
```

Constraints to validate data

Limit what values can be stored in a column

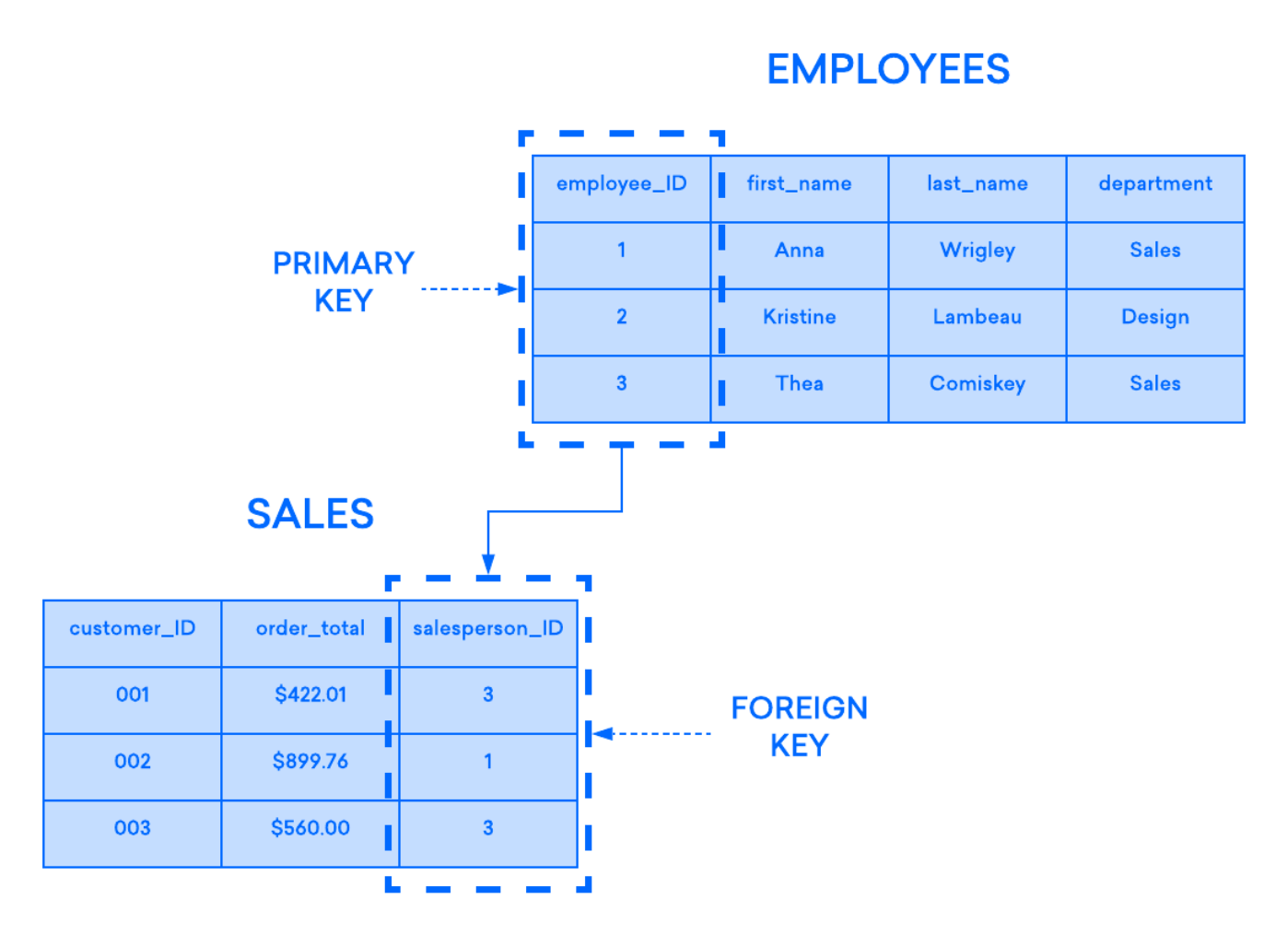
- `primary key` : `unique` & `not null`
  - `unique` : must be unique
  - `not null` : must have a value
- `foreign key` : references another table
- `check` : must satisfy the expression

  ...

---

https://www.sqlite.org/lang_createtable.html

`PRIMARY KEY` and `FOREIGN KEY` constraints



`PRIMARY KEY` and `FOREIGN KEY` constraints

```
CREATE TABLE EMPLOYEES (
    employee_ID numeric,
    first_name text,
    last_name text,
    department text,
    PRIMARY KEY (employee_ID)
);
CREATE TABLE SALES (
    customer_ID numeric,
    order_total numeric,
    salesperson_ID numeric,
    FOREIGN KEY (salesperson_ID) REFERENCES EMPLOYEES(employee_ID)
);
```

The `salesperson_ID` in `SALES` must match an `employee_ID` in `EMPLOYEES`

➡️ enforces referential integrity between the two tables

`INSERT INTO`: Insert data into a table

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

```
-- Insert a new book with only id and title
INSERT INTO books (id, title)
VALUES (1, 'Harry Potter and the Sorcerer''s Stone');
-- Insert a new book with all columns
INSERT INTO books
VALUES (1, 'Harry Potter and the Sorcerer''s Stone', 'J.K. Rowling', 1997);
-- Insert multiple books
INSERT INTO books (id, title, author, published_year)
VALUES
    (1, 'Harry Potter and the Sorcerer''s Stone', 'J.K. Rowling', 1997),
    (2, ...),
    (3, ...);
```

🤔 Rejected inserts

```
CREATE TABLE books (
    id numeric,
    title text,
    author text,
    published_year numeric,
    PRIMARY KEY (id),                  -- id is the primary key
    CHECK (published_year >= 1900) -- published_year must be >= 1900
);

INSERT INTO books
VALUES ('Harry Potter and the Sorcerer''s Stone');

INSERT INTO books (id, title)
VALUES ('Harry Potter and the Sorcerer''s Stone');

INSERT INTO books (id)
VALUES (1, 'Harry Potter and the Sorcerer''s Stone');
```

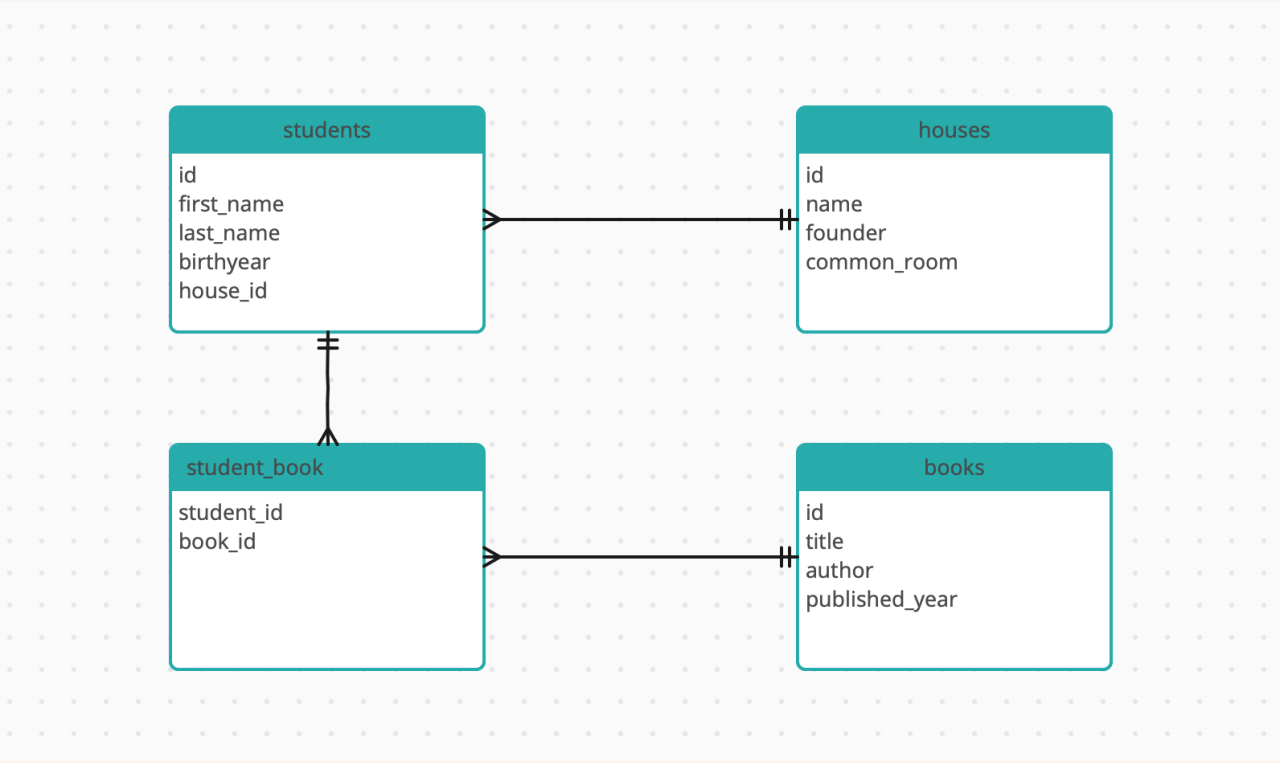
🤔 Rejected inserts

```
-- continued from previous slide

INSERT INTO books (id)
VALUES ('Harry Potter and the Sorcerer''s Stone');

INSERT INTO books (title)
VALUES ('Harry Potter and the Sorcerer''s Stone');

INSERT INTO books (id, title, author, published_year)
VALUES (1, 'Harry Potter and the Sorcerer''s Stone', 'J.K. Rowling', 1897);
```

Create student_book table

student_book:

- A student can appear across more than one book
- A book can have many students
- **Associative Entity** to represent many-to-many relationship between two entities



Primary Key?

Foreign Key?

Create student_book table

```
CREATE TABLE student_book (
    student_id int,
    book_id int,
    -- Composite PK: student_id and book_id together form a PK
    PRIMARY KEY (student_id, book_id),
    -- FK: student_id to refreence the students table
    FOREIGN KEY (student_id) REFERENCES students(id),
    -- FK: book_id to reference the books table
    FOREIGN KEY (book_id) REFERENCES books(id)
);
```

Insert data into books table (with column names)

```sql
-- Insert data into specific columns in the books table
INSERT INTO books (id, title, author, published_year) VALUES
-- each entry is separated by a comma ,
(201, 'Harry Potter and the Philosopher''s Stone', 'J.K. Rowling', 1997),
(202, 'Harry Potter and the Chamber of Secrets', 'J.K. Rowling', 1998),
(203, 'Harry Potter and the Prisoner of Azkaban', 'J.K. Rowling', 1999),
(204, 'Harry Potter and the Goblet of Fire', 'J.K. Rowling', 2000),
(205, 'Harry Potter and the Order of the Phoenix', 'J.K. Rowling', 2003),
(206, 'Harry Potter and the Half-Blood Prince', 'J.K. Rowling', 2005),
-- last entry without a comma
(207, 'Harry Potter and the Deathly Hallows', 'J.K. Rowling', 2007);
```

Insert data into student_book table (without column names)

```
INSERT INTO student_book VALUES
(101, 201), (101, 202), (101, 203), (102, 201), (102, 203),
(102, 204), (103, 201), (103, 204), (104, 202), (104, 205),
(105, 203), (105, 206), (106, 201), (106, 203), (106, 207),
(107, 205), (107, 206), (108, 204), (108, 206), (109, 203),
(109, 207), (110, 205), (111, 204), (112, 201), (113, 203),
(114, 202), (115, 205), (116, 204), (117, 201), (118, 202);
```

💻 Add Movies table to Harry Potter database

- Table name: `movies`
- Columns:
  - `id` int
  - `main_char_id` int
  - `title` text
  - `director` text
  - `year` int
- Constraints:
  - primary key: `id`
  - foreign key: `main_char_id` ➡️ `id` in `students` table

💻 Insert data into Movies table

```
1, 101, 'Harry Potter and the Sorcerer''s Stone', 'Chris Columbus', 2001
2, 101, 'Harry Potter and the Chamber of Secrets', 'Chris Columbus', 2002
3, 101, 'Harry Potter and the Prisoner of Azkaban', 'Alfonso Cuarón', 2004
4, 101, 'Harry Potter and the Goblet of Fire', 'Mike Newell', 2005
5, 101, 'Harry Potter and the Order of the Phoenix', 'David Yates', 2007
6, 101, 'Harry Potter and the Half-Blood Prince', 'David Yates', 2009
7, 101, 'Harry Potter and the Deathly Hallows – Part 1', 'David Yates', 2010
8, 101, 'Harry Potter and the Deathly Hallows – Part 2', 'David Yates', 2011
```

FK constraint violation example

Rejected: FK constraint violation (main_char_id 999 doesn't exist in students table)

```
INSERT INTO movies (id, main_char_id, title, director, year)
VALUES (9, 999, 'Harry Potter and the Cursed Child', 'John Doe', 2016);
```

DDL: `ALTER TABLE`

Add columns

```
ALTER TABLE mytable
ADD COLUMN column_name numeric;
```

Add constraints

```
ALTER TABLE mytable
ADD CONSTRAINT constraint_name
FOREIGN KEY (column_name) REFERENCES other_table(other_column);
```

DDL: `ALTER TABLE`

Remove columns

```
ALTER TABLE mytable
DROP COLUMN column_name;
```

Rename the table

```
ALTER TABLE mytable
RENAME TO new_table_name;
```

DDL: `DROP TABLE`

`DROP TABLE` removes a table and all its data

```
CREATE TABLE mytable (id int);
DROP TABLE mytable;                -- Drop mytable
DROP TABLE mytable;                -- Error because mytable doesn't exist anymore
```

`DROP TABLE IF EXISTS` removes a table only if it exists

```
CREATE TABLE mytable (id int);
DROP TABLE mytable;                  -- Drop mytable
DROP TABLE IF EXISTS mytable;        -- No error even if mytable doesn't exist
```

Useful when you want to recreate a table

```
CREATE TABLE mytable (id int);
CREATE TABLE mytable (id int);       -- Error because mytable already exists
DROP TABLE IF EXISTS mytable;        -- Drop mytable if it exists
CREATE TABLE mytable (id int);       -- Now this works
```

DML: `UPDATE` table

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

```
UPDATE books
SET published_year = 1998
WHERE id = 1;
```

DML: `DELETE FROM` table

```
DELETE FROM table_name
WHERE condition;
```

```
DELETE FROM books
WHERE id = 1;
```

💻 Update and delete data from the Movies table

1. Remove director column from the movies table
2. Add a new column `director_id` to the movies table
3. Delete the movie with id 1 from the movies table
4. Update the year of the movie with id 2 to 2003