```
--Create a new tbl with foreign key and unique constraint
CREATE TABLE `person_tbl` (
   `id` INT AUTO_INCREMENT PRIMARY KEY,
   `first_name` VARCHAR(255) NOT NULL,
   `last_name` VARCHAR(255) NOT NULL,
   `dob` DATE,
   CONSTRAINT full_name UNIQUE (first_name,last_name),
   FOREIGN KEY(`sid`) REFERENCES `student_tbl` (`id`)
) ENGINE=InnoDB;
```

```
--Add new entry
INSERT INTO client (first_name,last_name,dob,sid)
VALUES ('Sara','Smith','1970-01-02');
```

```
--Add new entry with foreign key
INSERT INTO project (cid,name,notes)
VALUES (
   (SELECT id FROM client
WHERE first_name='Sara' AND last_name='Smith'),
   'diamond',
   'Should be done by Jan 2017');
```

```
--find film with max length and min rental duration separate tbl
SELECT film_id FROM film
WHERE length = (SELECT MAX(length) FROM film AS max_len)
AND
rental_duration=(
   SELECT MIN(rental_duration) FROM film AS min_rent);
```

```
-- the number of times ed chases acts in each category
SELECT cate.name, COUNT(act.actor_id) FROM category cate
   LEFT JOIN film_category AS fc ON cate.category_id=fc.category_id
   LEFT JOIN film AS f ON fc.film_id=f.film_id
   LEFT JOIN film_actor AS fa ON f.film_id=fa.film_id
   LEFT JOIN actor AS act ON
      (fa.actor_id=act.actor_id) AND
      (act.first_name='ED' AND act.last_name='CHASE')
GROUP BY cate.category_id
ORDER BY cate.name
```

```
-- length of time each actor has starred in sci-fi movies
SELECT act.first_name,act.last_name, sci_fi.sum
FROM actor act
LEFT JOIN
   (SELECT act.actor_id AS actor_id, SUM(f.length)
   AS sum FROM actor act
      INNER JOIN film_actor fa ON act.actor_id=fa.actor_id
      INNER JOIN film f ON fa.film_id=f.film_id
      INNER JOIN film_category fc ON f.film_id=fc.film_id
      INNER JOIN category cate ON fc.category_id=cate.category_id
   WHERE cate.name='Sci-Fi'
   GROUP BY act.actor_id
) AS sci_fi ON act.actor_id=sci_fi.actor_id
```

```
-- find the actors who never starred in sci-fi films
SELECT act.first_name, act.last_name FROM actor act
WHERE act.actor_id NOT IN
   (SELECT act.actor_id FROM actor act
      INNER JOIN film_actor fa ON  act.actor_id=fa.actor_id
      INNER JOIN film f ON fa.film_id=f.film_id
      INNER JOIN film_category fc ON f.film_id=fc.film_id
      INNER JOIN category cate ON fc.category_id=cate.category_id
WHERE cate.name='Sci-Fi')
GROUP BY act.actor_id;
```

```
-- joining two different queries into one
(SELECT f.title FROM film f
   INNER JOIN film_actor fa ON f.film_id=fa1.film_id
   INNER JOIN actor act ON fa1.actor_id=act1.actor_id
   INNER JOIN film_actor fa1 ON f.film_id=fa.film_id
   INNER JOIN actor act1 ON fa.actor_id=act.actor_id
WHERE (act.first_name='WARREN')
   AND (act1.first_name='KIRSTEN')
ORDER BY f.title DESC;
```

```
-- Other SQL
SELECT * FROM foo_tbl
DELETE FROM foo_tbl WHERE col=...;
UPDATE uni_student
SET firstname='Kevin', lastname='Turkington' WHERE id=1
```

```
-- Attribute Types
INT, BIGINT, FLOAT, DECIMAL, VARCHAR(255), TEXT,
DATE
```

```
-- Aggregate Functions
Works on multiple rows, Group by is used to group rows into sets
AVG(/* col or select */), SUM(/* col or select */), COUNT( )
```

```
-- Joins
INNER JOIN: intersection of two tables
UNION (full outer): combines two tables
LEFT JOIN: all records from the MAIN and matched from the
joining  tables.
RIGHT JOIN: all records from the JOINING and matched from the
MAIN tables.
JOIN: all from JOINING TO MAIN
```

```
-- Layers of Abstraction
External: front end user data must make sence
Conceptual: how data is organized for the database desiginer
Internal: Mappings to actual data on physical storage media
```

```
-- Helpful MySQL hints
-Delete can violate Referential integrity (foreign keys)
-Insert can violate Referential and domain integrity (data length or
null)
-Cascade delete for a foreign key + delete all corresponding data will
also be deleted
```

```
-- Schemas
exist for entities and relationships and are composed of attributes and constraints
```

```
--Integrity constraints
-Domain: restricted domain of an attribute (INT, VARCHAR, ETC)
-Key: requires that entries in a column combo must be unque
-Referential: requires that an attribute be present in another table (foreign key)
-Semantic: rules about the system outside of the database
-NOT NULL: requires a value to be specified
-Entity: primary key cannot be null
```

```
-- Database normalization
-reduction of redundant data
-functional dependencies if x determine y, y is functionally dependent on x. x-> y
- 1NF: ex. tuples fixed by dividing into seperate columns
- 2NF: Every non primary attribute must not be partially dependant on another col.
Ex. course name can be turned into a lookup table.
- 3NF: non primary attribute can be transitively dependant on any key.
Ex. role priority on a person and role on a person is redundant. Fixed by creating a
lookup table.
- BCNF: like 3NF but requires no attribute be transitively dependent on any key.
Ex. toppings table has meat(beef) and cheese(cheddar) those can be its own table
(meat table, cheese table).
```

```
-- Relational Algebra
```

$\sigma_{col=val}$: WHERE $\Pi_{col1, col2, col3}$: SELECT $\blacktriangleright\blacktriangleleft_{(col=col\ or\ val)}$: UNION

X- cross product AxB
∩- Intersection: must have identical domains
U - Union: will combine duplicate rows, must have identical domains
(-) - Difference: A-B, must have identical domains
σ - selection, filter rows
Π - projection, filter columns
Examples:

$$(Model \blacktriangleright\blacktriangleleft_{(Model.model\_id=Vehicle.fk\_model\_id)} Vehicle)$$

$$\Pi_{make.make\_name,\ model.model\_name}(\sigma_{year=1976}(Make \blacktriangleright\blacktriangleleft_{(Make.make\_id=Vehicle.fk\_make\_id)}(Model \blacktriangleright\blacktriangleleft_{(Model.model\_id=Vehicle.fk\_model\_id)} Vehicle)))$$

$$\sigma_{foo.a < 100}(foo \blacktriangleright\blacktriangleleft_{foo.b=bar.c} Barr)$$

MISSING ER DIAGRAM STUFF