|  |  |  |
| --- | --- | --- |
| --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  σcol=val: WHERE Πcol1, col2, col3: SELECT ▸◂(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 ▸◂(Model.model\_id=Vehicle.fk\_model\_id)Vehicle)  Πmake.make\_name, model.model\_name (  σyear=1976 (Make ▸◂(Make.make\_id=Vehicle.fk\_make\_id)(  Model ▸◂(Model.model\_id=Vehicle.fk\_model\_id)Vehicle)))  σfoo.a < 100 (foo ▸◂foo.b=bar.c Barr) |
|
|
|
|
|

MISSING ER DIAGRAM STUFF