5/8/24

* Super-Key:
(Combinations)

Candidate Key (subset of Super Key)

Attribute or set of attributes which identifies each row uniquely (CNIC & Date of Birth) (Combination

* Primary Key: subset of Candidate Key and Super Key.

Persons

Date of Birth	cnic	fname	Lname	passport	city	religion
14-5-1980	315-7896	Ali	Ahmed	R35147	Lahore	Muslim
14-5-1990	178-9167	Jeorge	Brown	R17876	Lahore	Christian
3-7-1995	189-7645	Bilal	shareef	R13843	Karachi	Muslim
17-7-1998	478-9715	Amna	Ashraf	R39561	Karachi	Muslim
⋮	⋮	⋮	⋮	⋮	⋮	⋮

* ✗ (Date of birth, fname, lname) (Not a candidate key)
- ✓ (cnic, religion)
- ✓ (cnic, fname, lname)
- ✗ (fname, lname, city)
- ✓ (cnic, city)
- ✓ (cnic)
- ✓ (cnic date of Birth)

: Candidate Key (Identifies uniquely)

✓: Candidate Key

✗: Not Candidate Key

* PK would be CNIC (Primary Key)

Foreign Key: Primary Key used in other table to make data consistent is called foreign key.

PK ↑

Courses

Course_code	Course_name	Course description	Course Credit Hours	Course Grading
CS-PF	Programming Fundamentals	PF with C#	3	A
IT-PF	Programming Fundamentals	PF with python	3	B
CS-DB	Database System	Database concept	3	C
IT-DB	Database system	Database core	3	D

FK ↑

Classes

Class_Code	Course_code	Room #
CS-PF-M	CS-PF	3
IT-PF-M	IT-PF	4
CS-DB-M	CS-DB	5
IT-DB-M	IT-DB	4

X IT-DBS (It not allows to add)

* Course code present in Courses table and classes table to make relation. PK of courses will be FK in classes.

* If any other course in added in classes as course code IT-DBS,

(Keys data)
Book page # 80, slide #106

it will not be added in it and shows error.

* Super-Key must uniquely identifies (concept Revised)
* Constraints of Database:
  (i) Primary Key constraints (Must be unique)
  (ii) Foreign Key constraints (not null)
* **Secondary Key:** used for retrieval of data from the table.
* Integrity of Data achiving means correctness of data:
  (i) Entity Integrity Rule (Entries unique)
  (ii) Referential Integrity Rule (May be null value or any reference table)

* **SQL → DDL**
  (Data definition language)
  → CREATE table
  → DROP table : Structure defines
  → ALTER table : change in schema
  - Add column, change column / Data type
    drop column, constraint, create/update/delete

④ Data manipulation (DML):

→ Insert, update, delete

⊙ Common SQL Datatypes: pg: 254 slide: 280

① Numeric:

→ Number(L,D) or Nomeric(L,D)

→ Integer

→ SmallInt : L: Length

→ Decimal(L,D) : D: Decimal

② Character:

→ Char(L)

→ VarChar(L) or varchar2(L)

③ Date:

→ DATE

⊙ CREATE Table:

```
CREATE TABLE tablename (
  column 1   data type   constraint,
  column 2   datatype    constraint
  PRIMARY KEY
  FOREIGN KEY
  CONSTRAINT );
```

→ Constraints:

① Primary Key (Unique or not null)
② Foreign Key
③ UNIQUE
④ NOT NULL

⑤ CREATE TABLE Courses (

course_code varchar(5) PRIMARY Key,
course_description varchar(100),
course_name varchar(50),
course_credithours SMALLINT
course_guideline varchar(500)
);

: It's not necessary that every table contains foreign key but Primary key is necessary.

→ CREATE TABLE Classes(

class_code varchar(5) PK,
course_code varchar(5),
Room_no INTEGER
FOREIGN KEY (course_code)
References courses
);

⊙ INSERT INTO TABLE:

→ INSERT INTO COURSES (define our order of insert)
(course_code, course_name,
course description, course_credit_
hours, course_guidelines (opt))
values ("CS-PF", "Programming Fundamentals",
"This is PF courses", 3, "ABC" (opt))

→ INSERT into classes values("CS-PF-4",
"CS-PF", 4); (without providing and following order)

⊙ UPDATE:

→ Update classes set Roomno=4
where class_code = "CS-PF-M"

: where is important otherwise applied in whole.

⊙ DELETE:

→ Delete from classes : table exists but rows deleted
where class_code="CS-PF-M".

: delete specific

⊙ Check:

Delete from courses
where course-code="IT-DB"

: (Not work because course-code is Primary Key)(we have to update in classes also to delete).

12/8/24

⊙ DDL:

Data Definition Language

-> CREATE Table / Constraint
-> DROP Table / constraint
-> ALTER Table / Constraint

⊛ Integrity Rules:

PRIMARY KEY DDL

FOREIGN KEY DML

⊛ Constraints:

Name assign

1. NOT NULL
2. UNIQUE
3. PRIMARY KEY
4. FOREIGN KEY
5. Check (Defined condition)
6. Default (value assigned by system)

(*) ALTER (change one or more columns) (structure change)
(Column data modify, Insert new one, drop one column).

(*) —o< (Maybe zero or multiple)
(zero or many)
—o| (zero or one).

(+) **Customer:**

```
CREATE TABLE customer(
    CUS_code Integer(number) PRIMARY KEY,
    CUS_lname varchar(50),
    CUS_fname varchar(50),
    CUS_initial varchar(50),
    CUS_areacode Integer,
    CUS_phone Integer,
    CUS_balance Integer
);
```

(starting and ending letters of name)

(varchar, char)

→
```
CREATE TABLE INVOICE(
    inv_number Integer PRIMARY KEY,
    inv_date Date,
    CUS_code Integer
```

: check (column IN ( ))

```
FOREIGN KEY (cus_code)
  references  customer
);
```

: (Constraints name assignment UNIQUE( ))

(CONSTRAINTS CUS_U11 UNIQUE(lname, fname))

(at last of CREATE DDL statement)

→ CREATE TABLE INVOICE_1(

```
INV_NUMBER  NUMBER PRIMARY KEY,
CUS_CODE    NUMBER,
INV_DATE    DATE DEFAULT SYSDATE,
FOREIGN KEY (CUS_CODE)
  REFERNCES CUSTOMER
);
```

→ CREATE TABLE LINES(

```
INV_NUMBER NUMBER PRIMARY KEY,
LINE_NUMBER NUMBER,
p_code NUMBER
);
```

→ CREATE TABLE XXZ(

```
FNAME varchar (15),
LNAME varchar (15),
PRIMARY KEY (FNAME,LNAME));
```

⊛ DB → Semester Project:

Deadline 15-08-24

1- Choose System
2- Enlist its requirements
3- Enlist its features
4- Design Database Model
5- Design Python App

→ CREATE TABLE LINES(

INV_NUMBER PK,

);

⊛ On-Update Cascade (Deleting courses is possible when updated in classes) (search it)

→ Create table lines(

inv_number varchar(50),
line_number varchar(50),
p-code varchar(50),
Line_units smallint,
Line_price smallint,
PRIMARY KEY (inv_number, line number)

```
FOREIGN KEY (p_code) References Product,
FOREIGN KEY (inv_number) References Invoice
);
```

19/8/24

* Lucid Charts (for ERD)

* JOIN:

To relate columns of two tables. (INNER, OUTER, LEFT, RIGHT and FULL)

Types.

1- INNER JOIN (only similar rows)

2- OUTER JOIN

- Left
- Right

Orders

order_id	customer_id	order_date
1	4	13-1-20
2	9	14-2-20
3	8	8-1-23
7	NULL	1-2-24
18	1	5-8-23

* Inner Join Eg.

```
->Select order_id, customer_id,
  customer_name, order_date
  from orders INNER JOIN
  customers on orders.customer_id=
        customers.customer_id.
  (This incorrect format)
(Not specified about
customer_id of which table)
```

PK

customer_id	customer_name	customer_city
1	Ali	Lah
2	Ahmad	Lah
4	Abdullah	Lah
9	Fahad	Kar
8	Amir	Isl
3		

Customers

* Left Join (First table record and also the common record of other table).

: select order_id, customer_id, customer_name from orders, customers where orders.customer_id = customers.customer_id
(without using join syntax)
(join one or more table by above syntax)

* select order_id, order_date, customer.fname, employees.lname from orders, customer, employee where orders.employee_id = employees.employee_id AND orders.customer_id = customer.customerid
: (Three tables joined)

id, date, fname of customer, lname of employee

* JOINS increases execution time (NOSQL solves this issue).
* Self-join (joins records of itself).
(<>) (Not equal operator).
* Group By (Aggregate function)
→ select max(salary) from employees group by deptno.

* Group By or Aggregate functions, if criteria required then we using HAVING statement.

* Query (1) (Display order details (id, product name, quantity, price).

Query (2) (Total amount of each order).

(i) Select order_id, product_name, product_quantity, unit_price from order details, products where order_details.product_id = products.product_id.

(ii) Select discount, orders.order_id from order_details, orders where order_details.order_id = orders.order_id

Next class Quiz

21/8/24

* Chapter # 8 pg: 349

* JOINS and Subquery:

→ Select → From

→ where → Having

→ Select * from table
where ... (subquery).

* Subquery can return:
  (i) One single value
  (ii) A list of values
  (iii) A virtual table

* Where clause in subquery
* Having clause in subquery.
* Any and all (Greater than all)
  (Greater than atleast one)
* Correlated subquery : QOH (Quantity on hand)
  will work as nested loop.
* Virtual table.

28/8/24

→ Query to display department wise count of employees.

emp_num	dept_num	emp_name	sal	hire_date

: select deptno, count(*) as count from emp group by dept_num

⊛ **View** → Virtual table : to repeat specific query.

: Create view DepartCountEmp As
(select dept_num, count(emp_no) as EmployCount from emp group by dept_num);

: Select * from DepartCountEmp

→ View will be updated as table updated later.

→ If primary key is identified then DML operations can be applied, otherwise not.

→ Security purpose or access limit to user can be provided by view. (Base table columns can be added into view, so limited access can be provided to user).

→ How to use DB with high-level language:

→ DB, DBMS } Oracle / MySQL

→ API / Package / Driver Program.
(source to communicate).

→ PyD:- (API) (Package - pymysql)
(in MySQL)

Steps:

1. Import the package that communicate with dbms.
2. create database connection
3. create cursor
4. Execute Query
5. Close connection & cursor

```
(*) function start ←  import pymysql
                      dbcon = None          : None (Null)
                      dbcursor = None
                      try :
                          dbcon = pymysql.connect
                          (host="localhost", User="ro
                          password = "123", database ="te
                          dbcursor = dbcon.cursor()
                          query = "select email, password
                                   from users"
                          db.cursor.execute(query)
                          row = dbcursor.fetchone()
```

∴ pycharm (IDE)
: MySQL CommandLine

```
row2 = dbcursor.fetchone()        : 1st method
rows = dbcursor.fetchALL()        . (for r in rows: print(r[0],r[1])
except Exception as e:
    print(str(e))
finally:
    dbcursor.close()   → if dbcursor!=None:
    dbcon.close()      → if dbcon!=None:
```

(4c) For App:

function getUserData.

-> Login (Email, Password)

-> def signup(email,password):

```
    // same as getUserData
    query = "Inerst into user values(%s,%s
    args = (email,password)        : "tuple".
    dbcursor.execute(query, args).
    dbcon.commit()                 : to save
    signup = True
    :
    finally :
    :
    return signup
email = input(" Enter your email to register")
password = input(" Enter your password to register")
```

signup = signup(email, password)

* def signin(email, password):
    dbcon = None
    dbCursor = None
    signin = False
    try:
        dbcon ....
        dbcursor = dbcon.cursor()
        query = "select email, password from users"
        dbcursor.execute(query)
        for r in dbcursor.fetchone():
            if(r[0]==email && r[1]==password)
            signin = True
    except Exception as e:
        str(e)
    finally:

*
    query = "select email, password from users where email='%s' and password='%s'
    args = (email, password)

29/8/24

* File Format and Permanent Data Storage. Updating Poorly Designed Databases.
* Composite Primary Key (PROJ_NUM,
* INF (Normalization) (ReDesign table with Proper Primary Key without any repeating group) (1st Normal Form) (Non-repeating group)
* Functional dependence (one key is delermined by other)
* Partial dependency (one key has not relational with other) (emp_name has direct relation emp_num not with complete primary key) (One key has relational with some keys of composite key).

→ To solve partial dependency issue, we make other table:

Proj_Num	Proj_Name

→ Project table, Employee table, Assignment table (It is 2NF).

④ Transitive dependency (Make non-key attributes into separate table and choose it from other table).

④ SQL:

Complete Chapter.

④ Syllabus:

→ Chapter: 1 complete
→ Chapter: 2 complete
→ Chapter: 3 complete
→ Chapter: 4 complete
→ Chapter: 6 till 3NF
→ Chapter: 7 complete
→ Chapter: 8 Till Views.

→ (conceptual paper) (NF, ERD, python etc.....)