Full-Stack Web Development

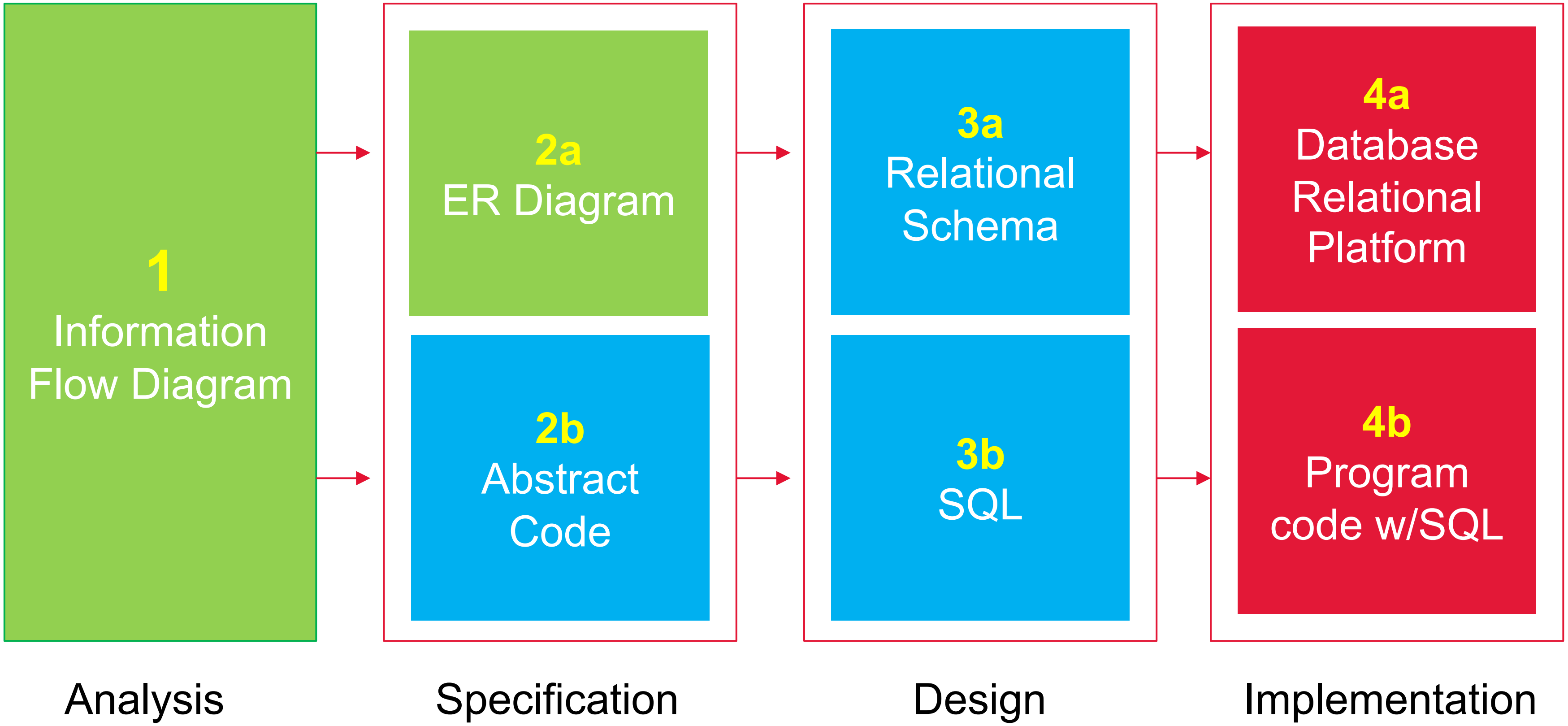
FS1030 – **Database Design and Principles**

school of continuing studies | YORK UNIVERSITÉ UNIVERSITY

What we learned so far

- Fundamentals of database
- Use cases around databases
- Why and when do we need various database types
- Data modeling and architecture
- Methodology
- Information flow diagrams
- Entity Relationship Diagrams

What we are going to learn



1
Information Flow Diagram

2a
ER Diagram

2b
Abstract Code

3a
Relational Schema

3b
SQL

4a
Database Relational Platform

4b
Program code w/SQL

Analysis

Specification

Design

Implementation

Example – Login Page

Abstract Code

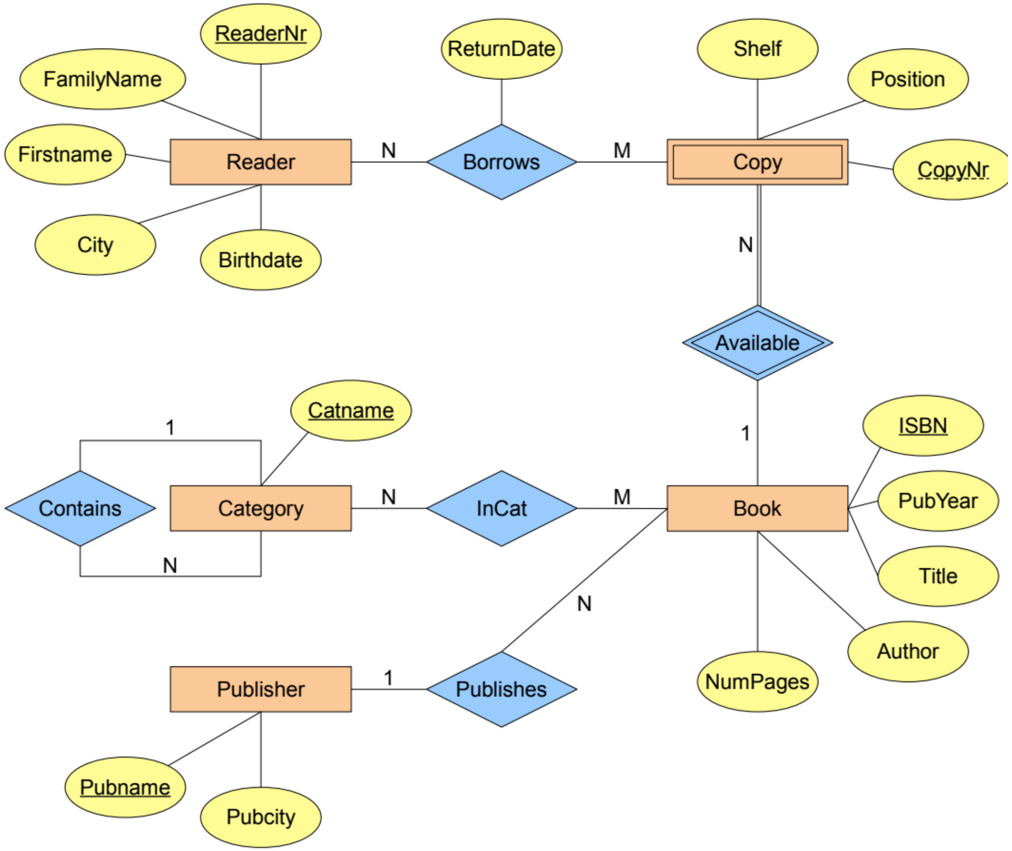
```
▪ User enters username (‘$Username’) and password (‘$Password’) input fields
▪ IF username is not empty AND password is not empty, AND Username does not contain invalid characters AND password does not contain any escape string THEN:
        ▪ When Login button is clicked:
                • IF user record is found but User.Password != ‘$Password’
                        o Go back to Login form with error message
                • ELSE:
                        o Store login information as session variable ‘$uid’
                        o IF User.role = ‘Administrator’
                                ▪ Store ‘yes’ in session variable ‘$isAdmin’
                        o ELSE
                                ▪ Store ‘no’ in session variable ‘$isAdmin’
                        o Go to User Menu page
▪ ELSE username or password field value is invalid, display Login form with error message
▪ When Register button is clicked:
        o Display Register
```

Group Activity

Assume there is a library system with the following properties.

- The library contains one or several copies of the same book.
- Every copy of a book has a copy number and is located at a specific location in a shelf.
- A copy is identified by the copy number and the ISBN number of the book.
- Every book has a unique ISBN, a publication year, a title, an author, and a number of pages.
- Books are published by publishers.
- A publisher has a name as well as a location.
- Within the library system, books are assigned to one or several categories.
- A category can be a subcategory of exactly one other category.
- A category has a name and no further properties.
- Each reader needs to provide his/her family name, his/her first name, his/her city, and his/her date of birth to register at the library.
- Each reader gets a unique reader number.
- Readers borrow copies of books.
- Upon borrowing the return date is stored.

Group Activity



1. Define the data formats needed

Attribute	Datatype	Nullable

2. Define the constraints
3. Write the abstract code for reader registration.

Keys

Primary Key – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.

Super Key – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.

Candidate Key – A super key with no redundant attribute is known as candidate key

Alternate Key – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.

Composite Key – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.

Foreign Key – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Primary Key

Primary Key – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.

Table name: Student

Stu_Id	Stu_Name	Stu_Age
101	Steve	23
102	John	24
103	Robert	28
104	Steve	29
105	Carl	29

- We denote usually denote it by underlining the attribute name (column name).
- The value of primary key should be unique for each row of the table. The column(s) that makes the key cannot contain duplicate values.
- The attribute(s) that is marked as primary key is not allowed to have null values.
- Primary keys are not necessarily to be a single attribute (column). It can be a set of more than one attributes (columns). For example {Stu_Id, Stu_Name} collectively can identify the tuple in the above table.

Keys

Super Key – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.

Table: Employee

Emp_SSN	Emp_Number	Emp_Name
123456789	226	Steve
999999321	227	Rogers
888997212	228	Ted
777778888	229	Robert

- {Emp_SSN}
- {Emp_Number}
- {Emp_SSN, Emp_Number}
- {Emp_SSN, Emp_Name}
- {Emp_SSN, Emp_Number, Emp_Name}
- {Emp_Number, Emp_Name}

Keys

Candidate Key – A super key with no redundant attribute is known as candidate key

Table: Employee

Emp_SSN	Emp_Number	Emp_Name
123456789	226	Steve
999999321	227	Rogers
888997212	228	Ted
777778888	229	Robert

- {Emp_Id} – No redundant attributes
- {Emp_Number} – No redundant attributes
- {Emp_Id, Emp_Number} – Redundant attribute. Either of those attributes can be a minimal super key as both of these columns have unique values.
- {Emp_Id, Emp_Name} – Redundant attribute Emp_Name.
- {Emp_Id, Emp_Number, Emp_Name} – Redundant attributes. Emp_Id or Emp_Number alone are sufficient enough to uniquely identify a row of Employee table.
- {Emp_Number, Emp_Name} – Redundant attribute Emp_Name.

Keys

Foreign Key – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Course_enrollment table:

Course_Id	Stu_Id
C01	101
C02	102
C03	101
C05	102
C06	103
C07	102

Student table:

Stu_Id	Stu_Name	Stu_Age
101	Chaitanya	22
102	Arya	26
103	Bran	25
104	Jon	21

the Stu_Id column in Course_enrollment table is a foreign key as it points to the primary key of the Student table.

Other definitions

Secondary or Alternative key
The candidate key which are not selected as primary key are known as secondary keys or alternative keys.

Non-key Attributes
Non-key attributes are the attributes or fields of a table, other than **candidate key** attributes/fields in a table.

Non-prime Attributes
Non-prime Attributes are attributes other than **Primary Key attribute(s).**

Dependencies

Functional Dependency
If the information stored in a table can uniquely determine another information in the same table, then it is called Functional Dependency.

Fully-Functional Dependency
An attribute is fully functional dependent on another attribute, if it is Functionally Dependent on that attribute and not on any of its proper subset.

Transitive Dependency
When an indirect relationship causes functional dependency. If P -> Q and Q -> R is true, then P-> R is a transitive dependency.

Multivalued Dependency
When existence of one or more rows in a table implies one or more other rows in the same table, then the Multi-valued dependencies occur.

Partial Dependency
Partial Dependency occurs when a nonprime attribute is functionally dependent on part of a candidate key.

Dependencies

Functional Dependency
If the information stored in a table can uniquely determine another information in the same table, then it is called Functional Dependency.

EmpID	**EmpName**	**EmpAge**
E01	Amit	28
E02	Rohit	31

In the above table, **EmpName** is functionally dependent on **EmpID** because **EmpName** can take only one value for the given value of **EmpID:**

EmpID -> EmpName

Dependencies

Fully-Functional Dependency
An attribute is fully functional dependent on another attribute, if it is Functionally Dependent on that attribute and not on any of its proper subset.

ProjectID	**ProjectCost**
001	1000
002	5000

EmpID	**ProjectID**	**Days** (spent on the project)
E099	001	320
E056	002	190

The above relations states:

EmpID, ProjectID, ProjectCost -> Days
However, it is not fully functional dependent.

Whereas the subset {**EmpID, ProjectID**} can easily determine the {**Days**} spent on the project by the employee.

This summarizes and gives our fully functional dependency:
{EmpID, ProjectID} -> (Days)

Dependencies

Transitive Dependency

When an indirect relationship causes functional dependency. If P -> Q and Q -> R is true, then P-> R is a transitive dependency.

Author_ID	Author	Book	Author_Nationality
Auth_001	Orson Scott Card	Ender's Game	United States
Auth_002	Margaret Atwood	The Handmaid's Tale	Canada

In the AUTHORS example above:
***Book* → *Author*:** Here, the *Book* attribute determines the *Author* attribute. If you know the book name, you can learn the author's name. However, *Author* does not determine *Book*, because an author can write multiple books. For example, just because we know the author's name Orson Scott Card, we still don't know the book name.

***Author* → *Author_Nationality*:** Likewise, the *Author* attribute determines the *Author_Nationality*, but not the other way around; just because we know the nationality does not mean we can determine the author.
But this table introduces a transitive dependency:

Book* →*Author_Nationality: If we know the book name, we can determine the nationality via the Author column.

Dependencies

Partial Dependency

Partial Dependency occurs when a nonprime attribute is functionally dependent on part of a candidate key.

StudentID	**ProjectNo**	**StudentName**	**ProjectName**
S01	199	Katie	Geo Location
S02	120	Ollie	Cluster Exploration

The prime key attributes are **StudentID** and **ProjectNo.**

As stated, the non-prime attributes i.e. **StudentName** and **ProjectName** should be functionally dependent on part of a candidate key, to be Partial Dependent.

The **StudentName** can be determined by **StudentID** that makes the relation Partial Dependent.

The **ProjectName** can be determined by **ProjectNo**, which makes the relation Partial Dependent.

Anamolies in database

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee’s id, emp_name for storing employee’s name, emp_address for storing employee’s address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Toronto	D001
101	Rick	Toronto	D002
123	Maggie	Vancouver	D890
166	Glenn	Montreal	D900
166	Glenn	Montreal	D004

Anamolies in database

emp_id	emp_name	emp_address	emp_dept
101	Rick	Toronto	D001
101	Rick	Toronto	D002
123	Maggie	Vancouver	D890
166	Glenn	Montreal	D900
166	Glenn	Montreal	D004

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows. If the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

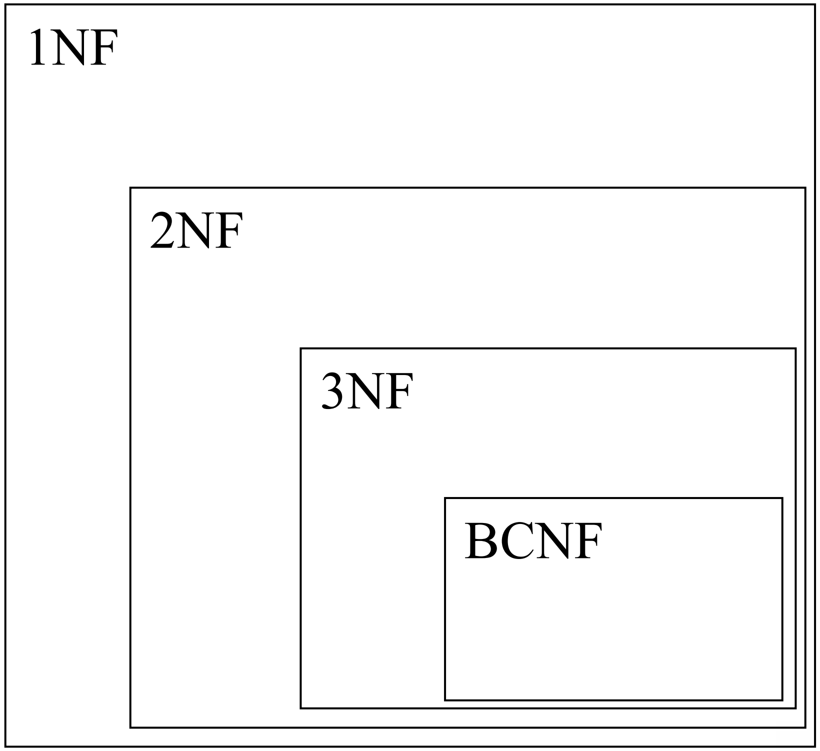
Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

Solution: Normalization

Normalization presents a set of rules that tables and databases must follow to be well structured. Normalization is used for:

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e. data is logically stored.



a relation in BCNF, is also in 3NF

a relation in 3NF is also in 2NF

a relation in 2NF is also in 1NF

1NF

A table is in the first normal form if

- The domain of each attribute contains only ***atomic values***, and
- The value of each attribute contains only a ***single value*** from that domain.

In layman's terms. it means every column of your table should only contain *single values*

Example – 1NF

For a library

Patron ID	Borrowed books
C45	B33, B44, B55
C12	B56

Patron ID	Borrowed book
C45	B33
C45	B44
C45	B33
C12	B56

Solve it for 1NF

Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	Toronto	8912312390
102	Jon	New York	8812121212 9900012222
103	Ron	Boston	7778881212
104	Lester	Winnipeg	9990000123 8123450987

Solve it for 1NF

ID	Name	Courses
1	A	C1,C2
2	E	C3
3	M	C2,C3

2NF

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No partial dependency.

An attribute that is not part of any candidate key is known as non-prime attribute.

2NF

BookNo	Patron	PhoneNo
B3	J. Fisher	555-1234
B2	J. Fisher	555-1234
B2	M. Amer	555-4321

Candidate key is {BookNo, Patron}

Patron → PhoneNo

BookNo	Patron
B3	J. Fisher
B2	J. Fisher
B2	M. Amer

Patron	PhoneNo
J. Fisher	555-1234
M. Amer	555-4321

Solve it for 2NF

Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Solve it for 2NF

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non prime attribute: teacher_age

teacher_id	teacher_age
111	38
222	38
333	40

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

3NF

A table is in 3NF if

- it is in 2NF and
- No transitive dependency

BookNo	Patron	Address	Due
B1	J. Fisher	101 Main Street	3/2/15
B2	L. Perez	202 Market Street	2/28/15

- ❑ Candidate key is BookNo
- ❑ Patron → Address

BookNo	Patron	Due
B1	J. Fisher	3/2/15
B2	L. Perez	2/28/15

Patron	Address
J. Fisher	101 Main Street
L. Perez	202 Market Street

Solve it for 3NF

Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	ON	Toronto	Toronto
1002	Ajeet	222008	SK	Regina	Regina
1006	Lora	282007	SK	Regina	Downtown
1101	Lilly	292008	BC	Vancouver	Downtown
1201	Steve	222999	AB	Edmonton	MidCircle

Solve it for 3NF

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	ON	Toronto	Lawrence
1002	Ajeet	222008	SK	Regina	St. Pleasant
1006	Lora	282007	SK	Regina	Lake
1101	Lilly	292008	BC	Vancouver	Downtown
1201	Steve	222999	AB	Edmonton	Uptown

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on
Candidate Keys: {emp_id}
Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Solve it for 3NF

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	ON	Toronto	Lawrence
1002	Ajeet	222008	SK	Regina	St. Pleasant
1006	Lora	282007	SK	Regina	Lake
1101	Lilly	292008	BC	Vancouver	Downtown
1201	Steve	222999	AB	Edmonton	Uptown

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

emp_zip	emp_state	emp_city	emp_district
282005	ON	Toronto	Lawrence
222008	SK	Regina	St. Pleasant
282007	SK	Regina	Lake
292008	BC	Vancouver	Downtown
222999	AB	Edmonton	Uptown

BCNF (Boyce-Codd Normal Form)

- Stricter form of 3NF
- That for a dependency A → B, A cannot be a **non-prime attribute**, if B is a **prime attribute**
- Most tables that are in 3NF also are in BCNF

Manager	Project	Branch
Alice	Alpha	Austin
Alice	Delta	Austin
Carol	Alpha	Houston
Dean	Delta	Houston

□ Manager → Branch

□ {Project, Branch} → Manager

Manager	Project
Alice	Alpha
Bob	Delta
Carol	Alpha
Alice	Delta
Dean	Delta

Manager	Branch
Alice	Austin
Bob	Houston
Carol	Houston
Dean	Houston

Solve it for BCNF

Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Solve it for BCNF

Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}
The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

Solve it for BCNF

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}
This is now in BCNF as in both the functional dependencies left side part is a key.

4NF

- It should be in the **Boyce-Codd Normal Form**.
- And, the table should not have any **Multi-valued Dependency**.

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
1	Science	Hockey
1	Maths	Cricket

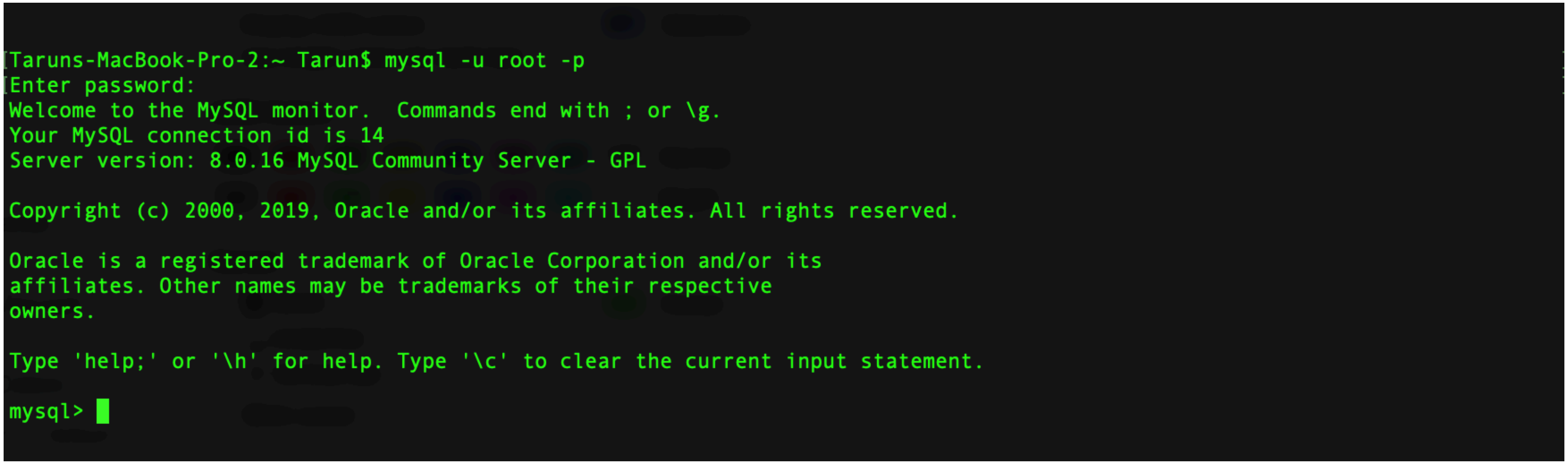
4NF

- It should be in the **Boyce-Codd Normal Form**.
- And, the table should not have any **Multi-valued Dependency**.

s_id	**course**
1	Science
1	Maths
2	C#
2	Php

s_id	**hobby**
1	Cricket
1	Hockey
2	Cricket
2	Hockey

MySQL Queries – MySQL command line



```
[Taruns-MacBook-Pro-2:~ Tarun$ mysql -u root -p
[Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.16 MySQL Community Server - GPL

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

MySQL Queries – MySQL command line

Creating and Deleting a Database

```
mysql> CREATE DATABASE fs1030;
Query OK, 1 row affected (0.03 sec)

mysql> DROP DATABASE fs1030;
Query OK, 0 rows affected (0.11 sec)

mysql> CREATE DATABASE IF NOT EXISTS fs1030;
Query OK, 1 row affected (0.01 sec)

mysql> DROP DATABASE IF EXISTS fs1030;
Query OK, 0 rows affected (0.00 sec)
```

MySQL Queries – MySQL command line

SHOW CREATE DATABASE

```
mysql> CREATE DATABASE IF NOT EXISTS fs1030;

mysql> SHOW CREATE DATABASE fs1030 \G
```

MySQL Queries – MySQL command line

USE DATABASE

```
mysql> USE fs1030;

mysql> SELECT DATABASE();

-- Show all the tables in the current database.
-- “fs1030” has no table (empty set).

mysql> SHOW TABLES;
Empty set (0.00 sec)
```

MySQL Queries – MySQL command line

CREATE TABLE

```
mysql> CREATE TABLE IF NOT EXISTS products (
         productID    INT UNSIGNED  NOT NULL AUTO_INCREMENT,
         productCode  CHAR(3)       NOT NULL DEFAULT '',
         name         VARCHAR(30)   NOT NULL DEFAULT '',
         quantity     INT UNSIGNED  NOT NULL DEFAULT 0,
         price        DECIMAL(7,2)  NOT NULL DEFAULT 99999.99,
         PRIMARY KEY  (productID)
       );
Query OK, 0 rows affected (0.08 sec)

-- Show all the tables to confirm that the "products" table has been created
mysql> SHOW TABLES;

mysql> SHOW CREATE TABLE products \G
```

MySQL Queries – MySQL command line

CREATE TABLE

```
mysql> CREATE TABLE IF NOT EXISTS products (
         productID    INT UNSIGNED  NOT NULL AUTO_INCREMENT,
         productCode  CHAR(3)       NOT NULL DEFAULT '',
         name         VARCHAR(30)   NOT NULL DEFAULT '',
         quantity     INT UNSIGNED  NOT NULL DEFAULT 0,
         price        DECIMAL(7,2)  NOT NULL DEFAULT 99999.99,
         PRIMARY KEY  (productID)
       );
Query OK, 0 rows affected (0.08 sec)

-- Show all the tables to confirm that the "products" table has been created
mysql> SHOW TABLES;

mysql> SHOW CREATE TABLE products \G
```

MySQL Queries – MySQL command line

INSERTING ROWS

INSERT INTO *tableName* (*column1Name*, ..., *columnNName*) VALUES (*row1column1Value*, ..., *row2ColumnNValue*), (*row2column1Value*, ..., *row2ColumnNValue*), ...

```
-- Insert a row with all the column values
mysql> INSERT INTO products VALUES (1001, 'PEN', 'Pen Red', 5000, 1.23);

-- Insert multiple rows in one command
-- Inserting NULL to the auto_increment column results in max_value + 1
mysql> INSERT INTO products VALUES
       (NULL, 'PEN', 'Pen Blue',  8000, 1.25),
       (NULL, 'PEN', 'Pen Black', 2000, 1.25);

-- Insert value to selected columns
-- Missing value for the auto_increment column also results in max_value + 1
mysql> INSERT INTO products (productCode, name, quantity, price) VALUES
       ('PEC', 'Pencil 2B', 10000, 0.48),
       ('PEC', 'Pencil 2H', 8000, 0.49);

-- Missing columns get their default values
mysql> INSERT INTO products (productCode, name) VALUES ('PEC', 'Pencil HB');

-- 2nd column (productCode) is defined to be NOT NULL
mysql> INSERT INTO products values (NULL, NULL, NULL, NULL, NULL);
```

MySQL Queries – MySQL command line

SELECT

SELECT *column1Name*, *column2Name*, ... FROM *tableName*

```
-- List all rows for the specified columns
mysql> SELECT name, price FROM products;

-- List all rows of ALL the columns. The wildcard * denotes ALL columns
mysql> SELECT * FROM products;

mysql> SELECT 1+1;

-- Multiple columns
mysql> SELECT 1+1, NOW();
```

MySQL Queries – MySQL command line

SELECT – Comparison Operator

SELECT *column1Name*, *column2Name*, ... FROM *tableName*

```
mysql> SELECT name, price FROM products WHERE price < 1.0;

mysql> SELECT name, quantity FROM products WHERE quantity <= 2000;

mysql> SELECT name, price FROM products WHERE productCode = 'PEN’;
-- String values are quoted
```

MySQL Queries – MySQL command line

SELECT – Pattern Matching

- 'abc%' matches strings beginning with 'abc';
- '%xyz' matches strings ending with 'xyz';
- '%aaa%' matches strings containing 'aaa';
- '___' matches strings containing exactly three characters;
- 'a_b%' matches strings beginning with 'a', followed by any single character, followed by 'b', followed by zero or more characters.

```
-- "name" begins with 'PENCIL’
mysql> SELECT name, price FROM products WHERE name LIKE 'PENCIL%’;

-- "name" begins with 'P', followed by any two characters,
-- followed by space, followed by zero or more characters
mysql> SELECT name, price FROM products WHERE name LIKE 'P__ %';
```

MySQL Queries – MySQL command line

SELECT – Arithmetic Operators(+ , -, *, /, DIV, %) and Logical Operators (AND, OR, NOT, XOR)

```
mysql> SELECT * FROM products WHERE quantity >= 5000 AND name LIKE 'Pen %’;
```

```
mysql> SELECT * FROM products WHERE quantity >= 5000 AND price < 1.24 AND
name LIKE 'Pen %’;
```

```
mysql> SELECT * FROM products WHERE NOT (quantity >= 5000 AND name LIKE
'Pen %’);
```

IN, NOT IN

```
mysql> SELECT * FROM products WHERE name IN ('Pen Red', 'Pen Black’);
```

BETWEEN, NOT BETWEEN

```
mysql> SELECT * FROM products WHERE (price BETWEEN 1.0 AND 2.0) AND
(quantity BETWEEN 1000 AND 2000);
```

MySQL Queries – MySQL command line

IS NULL, IS NOT NULL

```
mysql> SELECT * FROM products WHERE productCode IS NULL;

-- What happens when you run the below query?
mysql> SELECT * FROM products WHERE productCode = NULL;
```

ORDER BY

```
-- Order the results by price in descending order
mysql> SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC;

-- Order by price in descending order, followed by quantity in ascending (default) order
mysql> SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC, quantity;
```

MySQL Queries – MySQL command line

LIMIT

```
-- Display the first two rows
mysql> SELECT * FROM products ORDER BY price LIMIT 2;

-- Skip the first two rows and display the next 1 row
mysql> SELECT * FROM products ORDER BY price LIMIT 2, 1;
```

AS - Alias

```
mysql> SELECT productID AS ID, productCode AS Code,
          name AS Description, price AS `Unit Price`  -- Define aliases to be used as
display names
    FROM products
    ORDER BY ID;  -- Use alias ID as reference
```

Function CONCAT()

```
mysql> SELECT CONCAT(productCode, ' - ', name) AS `Product Description`, price
FROM products;
```

MySQL Queries – MySQL command line

DISTINCT

```
-- Without DISTINCT
mysql> SELECT price FROM products;

-- With DISTINCT on price
mysql> SELECT DISTINCT price AS `Distinct Price` FROM products;

-- DISTINCT combination of price and name
mysql> SELECT DISTINCT price, name FROM products;
```

GROUP BY Clause

```
mysql> SELECT * FROM products ORDER BY productCode, productID;

mysql> SELECT * FROM products GROUP BY productCode;
-- Only first record in each group is shown
```

MySQL Queries – MySQL command line

AGGREGATE FUNCTIONS - COUNT, MAX, MIN, AVG, SUM, STD, GROUP_CONCAT

```
-- Function COUNT(*) returns the number of rows selected
mysql> SELECT COUNT(*) AS `Count` FROM products; -- All rows without GROUP BY clause

mysql> SELECT productCode, COUNT(*) FROM products GROUP BY productCode;

-- Order by COUNT - need to define an alias to be used as reference
mysql> SELECT productCode, COUNT(*) AS count FROM products GROUP BY productCode ORDER BY count DESC;

mysql> SELECT MAX(price), MIN(price), AVG(price), STD(price), SUM(quantity) FROM products;
-- Without GROUP BY - All rows

mysql> SELECT productCode, MAX(price) AS `Highest Price`, MIN(price) AS `Lowest Price` FROM products GROUP BY productCode;
```

MySQL Queries – MySQL command line

AGGREGATE FUNCTIONS - COUNT, MAX, MIN, AVG, SUM, STD, GROUP_CONCAT

```
mysql> SELECT productCode, MAX(price), MIN(price),
              CAST(AVG(price) AS DECIMAL(7,2)) AS `Average`,
              CAST(STD(price) AS DECIMAL(7,2)) AS `Std Dev`,
              SUM(quantity)
       FROM products
       GROUP BY productCode;
       -- Use CAST(... AS ...) function to format floating-point numbers
```

HAVING CLAUSE

```
mysql> SELECT
          productCode AS `Product Code`,
          COUNT(*) AS `Count`,
          CAST(AVG(price) AS DECIMAL(7,2)) AS `Average`
       FROM products
       GROUP BY productCode
       HAVING Count >=3;
          -- CANNOT use WHERE count >= 3
```

HAVING is similar to WHERE, but it can operate on the GROUP BY aggregate functions; whereas WHERE operates only on columns.

MySQL Queries – MySQL command line

WITH ROLL UP

```
mysql> SELECT
           productCode,
           MAX(price),
           MIN(price),
           CAST(AVG(price) AS DECIMAL(7,2)) AS `Average`,
           SUM(quantity)
        FROM products
        GROUP BY productCode
        WITH ROLLUP;        -- Apply aggregate functions to all groups
```

MySQL Queries – MySQL command line

UPDATE

UPDATE *tableName* SET *columnName* = {*value*|NULL|DEFAULT}, ... WHERE *criteria*

```
-- Increase the price by 10% for all products
mysql> UPDATE products SET price = price * 1.1;

mysql> SELECT * FROM products;

-- Modify selected rows
mysql> UPDATE products SET quantity = quantity - 100 WHERE name = 'Pen Red’;

-- You can modify more than one values
mysql> UPDATE products SET quantity = quantity + 50, price = 1.23 WHERE name =
'Pen Red’;

mysql> SELECT * FROM products WHERE name = 'Pen Red';
```

CAUTION: If the WHERE clause is omitted in the UPDATE command, ALL ROWS will be updated.

MySQL Queries – MySQL command line

DELETE FROM

Use with extreme care! Records are NOT recoverable!!!
DELETE FROM *tableName* WHERE *criteria*

```
mysql> DELETE FROM products WHERE name LIKE 'Pencil%’;

-- Use this with extreme care, as the deleted records are irrecoverable!
mysql> DELETE FROM products;

mysql> SELECT * FROM products;
```

MySQL Queries – MySQL command line

LOADING FROM A FILE

Create a new file called products_in.csv with following data and save it under "d:\myProject" (for Windows) or "Documents" (for Mac)

```
\N,PEC,Pencil 3B,500,0.52
\N,PEC,Pencil 4B,200,0.62
\N,PEC,Pencil 5B,100,0.73
\N,PEC,Pencil 6B,500,0.47
```

(For Windows)

```
-- Need to use forward-slash (instead of back-slash) as directory separator
mysql> LOAD DATA LOCAL INFILE 'd:/myProject/products_in.csv' INTO TABLE
products COLUMNS TERMINATED BY ',' LINES TERMINATED BY '\r\n’;
```

(For Mac)

```
mysql> LOAD DATA LOCAL INFILE '~/Documents/products_in.csv' INTO TABLE
products COLUMNS TERMINATED BY ',';
```

MySQL Queries – MySQL command line

LOADING FROM A FILE

Create a new file called products_in.csv with following data and save it under "d:\myProject" (for Windows) or "Documents" (for Mac)

```
\N,PEC,Pencil 3B,500,0.52
\N,PEC,Pencil 4B,200,0.62
\N,PEC,Pencil 5B,100,0.73
\N,PEC,Pencil 6B,500,0.47
```

(For Windows)

```
-- Need to use forward-slash (instead of back-slash) as directory separator
mysql> LOAD DATA LOCAL INFILE 'd:/myProject/products_in.csv' INTO TABLE
products COLUMNS TERMINATED BY ',' LINES TERMINATED BY '\r\n’;
```

(For Mac)

```
mysql> LOAD DATA LOCAL INFILE '~/Documents/products_in.csv' INTO TABLE
products COLUMNS TERMINATED BY ',’;
```

- The default line delimiter (or end-of-line) is '\n' (Unix-style).
- If the text file is prepared in Windows, you need to include LINES TERMINATED BY '\r\n'.
- The default column delimiter is "tab" (in a so-called TSV file - Tab-Separated Values). If you use another delimiter, e.g. ',', include COLUMNS TERMINATED BY ','.
- You need to use \N for NULL.

MySQL Queries – MySQL command line

Mysqlimport

> **mysqlimport -u *username* -p --local *databaseName tableName*.tsv**

-- The raw data must be kept in a TSV (Tab-Separated Values) file with filename the same as tablename

```
-- EXAMPLES
-- Create a new file called "products.tsv" containing the following record,
--  and saved under "d:\myProject" (for Windows) or "Documents" (for Mac)
-- The values are separated by tab (not spaces).
\N  PEC  Pencil 3B  500  0.52
\N  PEC  Pencil 4B  200  0.62
\N  PEC  Pencil 5B  100  0.73
\N  PEC  Pencil 6B  500  0.47
```

(For Windows)

```
> cd path-to-mysql-bin
> mysqlimport -u root -p --local southwind d:/myProject/products.tsv
```

(For Macs)

```
$ cd /usr/local/mysql/bin
$ ./mysqlimport -u root -p --local southwind ~/Documents/products.tsv
```

MySQL Queries – MySQL command line

SELECT INTO OUTFILE

(For Windows)

```
mysql> SELECT * FROM products INTO OUTFILE 'd:/myProject/products_out.csv'
        COLUMNS TERMINATED BY ','
        LINES TERMINATED BY '\r\n';
```

(For Macs)

```
mysql> SELECT * FROM products INTO OUTFILE '~/Documents/products_out.csv'
        COLUMNS TERMINATED BY ',';
```

MySQL Queries – MySQL command line

RUNNING A SQL SCRIPT

```
DELETE FROM products;
INSERT INTO products VALUES (2001, 'PEC', 'Pencil 3B', 500, 0.52),
                            (NULL, 'PEC', 'Pencil 4B', 200, 0.62),
                            (NULL, 'PEC', 'Pencil 5B', 100, 0.73),
                            (NULL, 'PEC', 'Pencil 6B', 500, 0.47);
SELECT * FROM products;
```

- Save the above script in a file called load_products.sql under "d:\myProject" (for Windows) or "Documents" (for Mac).

(For Windows)

```
mysql> source d:/myProject/load_products.sql
   -- Use Unix-style forward slash (/) as directory separator
```

(For Macs)

```
mysql> source ~/Documents/load_products.sql
```

(For Windows)

```
> cd path-to-mysql-bin
> mysql -u root -p southwind < d:\myProject\load_products.sql
```

(For Macs)

```
$ cd /usr/local/mysql/bin
$ ./mysql -u root -p southwind < ~\Documents\load_products.sql
```

MySQL Queries – MySQL command line

One-To-Many Relationship

```
mysql> DROP TABLE IF EXISTS suppliers;

mysql> CREATE TABLE suppliers (
         supplierID  INT UNSIGNED  NOT NULL AUTO_INCREMENT,
         name        VARCHAR(30)   NOT NULL DEFAULT ",
         phone       CHAR(8)       NOT NULL DEFAULT ",
         PRIMARY KEY (supplierID)
       );

mysql> INSERT INTO suppliers VALUE
          (501, 'ABC Traders', '88881111'),
          (502, 'XYZ Company', '88882222'),
          (503, 'QQ Corp', '88883333’);
```

Alter Table

```
mysql> ALTER TABLE products
      ADD COLUMN supplierID INT UNSIGNED NOT NULL;
```

MySQL Queries – MySQL command line

SELECT with JOIN

```
mysql> SELECT products.name, price, suppliers.name
       FROM products
         JOIN suppliers ON products.supplierID = suppliers.supplierID
       WHERE price < 0.6;

-- Join via WHERE clause (lagacy and not recommended)
mysql> SELECT products.name, price, suppliers.name
       FROM products, suppliers
       WHERE products.supplierID = suppliers.supplierID
         AND price < 0.6;

-- Use aliases for column names for display
mysql> SELECT products.name AS `Product Name`, price, suppliers.name AS `Supplier Name`
       FROM products
         JOIN suppliers ON products.supplierID = suppliers.supplierID
       WHERE price < 0.6;

-- Use aliases for table names too
mysql> SELECT p.name AS `Product Name`, p.price, s.name AS `Supplier Name`
       FROM products AS p
         JOIN suppliers AS s ON p.supplierID = s.supplierID
       WHERE p.price < 0.6;
```

MySQL Queries – MySQL command line

Many to Many Relationships

```
mysql> CREATE TABLE products_suppliers (
         productID   INT UNSIGNED  NOT NULL,
         supplierID  INT UNSIGNED  NOT NULL,
                     -- Same data types as the parent tables
         PRIMARY KEY (productID, supplierID),
                     -- uniqueness
         FOREIGN KEY (productID)  REFERENCES products  (productID),
         FOREIGN KEY (supplierID) REFERENCES suppliers (supplierID)
       );

mysql> INSERT INTO products_suppliers VALUES (2001, 501), (2002, 501),
       (2003, 501), (2004, 502), (2001, 503);
-- Values in the foreign-key columns (of the child table) must match
--   valid values in the columns they reference (of the parent table)

mysql> ALTER TABLE products DROP FOREIGN KEY products_ibfk_1;

mysql> ALTER TABLE products DROP supplierID;

mysql> SELECT products.name AS `Product Name`, price, suppliers.name AS `Supplier Name`
       FROM products_suppliers
         JOIN products  ON products_suppliers.productID = products.productID
         JOIN suppliers ON products_suppliers.supplierID = suppliers.supplierID
       WHERE price < 0.6;
```

MySQL Queries – MySQL command line

Many to Many Relationships

```
mysql> CREATE TABLE products_suppliers (
         productID   INT UNSIGNED  NOT NULL,
         supplierID  INT UNSIGNED  NOT NULL,
                     -- Same data types as the parent tables
         PRIMARY KEY (productID, supplierID),
                     -- uniqueness
         FOREIGN KEY (productID)  REFERENCES products  (productID),
         FOREIGN KEY (supplierID) REFERENCES suppliers (supplierID)
       );

mysql> INSERT INTO products_suppliers VALUES (2001, 501), (2002, 501),
       (2003, 501), (2004, 502), (2001, 503);
-- Values in the foreign-key columns (of the child table) must match
--   valid values in the columns they reference (of the parent table)

mysql> ALTER TABLE products DROP FOREIGN KEY products_ibfk_1;

mysql> ALTER TABLE products DROP supplierID;

mysql> SELECT products.name AS `Product Name`, price, suppliers.name AS `Supplier Name`
       FROM products_suppliers
         JOIN products  ON products_suppliers.productID = products.productID
         JOIN suppliers ON products_suppliers.supplierID = suppliers.supplierID
       WHERE price < 0.6;
```

MySQL Queries – MySQL command line

Indexes

In MySQL, indexes can be built on:

- a single column (column-index)
- a set of columns (concatenated-index)
- on unique-value column (UNIQUE INDEX or UNIQUE KEY)
- on a prefix of a column for strings (VARCHAR or CHAR), e.g., first 5 characters.

```
mysql> CREATE TABLE employees (
       emp_no      INT UNSIGNED   NOT NULL AUTO_INCREMENT,
       name        VARCHAR(50)    NOT NULL,
       gender      ENUM ('M','F') NOT NULL,
       birth_date  DATE           NOT NULL,
       hire_date   DATE           NOT NULL,
       PRIMARY KEY (emp_no)  -- Index built automatically on primary-key column
    );
```

```
mysql> SHOW INDEX FROM employees \G
```

MySQL Queries – MySQL command line

Indexes

```
mysql> CREATE TABLE departments (
         dept_no    CHAR(4)      NOT NULL,
         dept_name  VARCHAR(40)  NOT NULL,
         PRIMARY KEY  (dept_no),   -- Index built automatically on primary-key column
         UNIQUE INDEX (dept_name)  -- Build INDEX on this unique-value column
       );

mysql> SHOW INDEX FROM departments \G

-- Many-to-many junction table between employees and departments
mysql> CREATE TABLE dept_emp (
         emp_no     INT UNSIGNED  NOT NULL,
         dept_no    CHAR(4)       NOT NULL,
         from_date  DATE          NOT NULL,
         to_date    DATE          NOT NULL,
         INDEX      (emp_no),         -- Build INDEX on this non-unique-value column
         INDEX      (dept_no),        -- Build INDEX on this non-unique-value column
         FOREIGN KEY (emp_no)  REFERENCES employees (emp_no)
           ON DELETE CASCADE ON UPDATE CASCADE,
         FOREIGN KEY (dept_no) REFERENCES departments (dept_no)
           ON DELETE CASCADE ON UPDATE CASCADE,
         PRIMARY KEY (emp_no, dept_no)  -- Index built automatically
      );
mysql> DESCRIBE dept_emp;
```

MySQL Queries – MySQL command line

More on Joins – Inner Join

```
mysql> DROP TABLE IF EXISTS t1, t2;

mysql> CREATE TABLE t1 (
        id      INT PRIMARY KEY,
        `desc`  VARCHAR(30)
      );
-- `desc` is a reserved word - must be back-quoted

mysql> CREATE TABLE t2 (
        id      INT PRIMARY KEY,
        `desc`  VARCHAR(30)
      );

mysql> INSERT INTO t1 VALUES
       (1, 'ID 1 in t1'),
       (2, 'ID 2 in t1'),
       (3, 'ID 3 in t1');

mysql> INSERT INTO t2 VALUES
       (2, 'ID 2 in t2'),
       (3, 'ID 3 in t2'),
       (4, 'ID 4 in t2');
```

MySQL Queries – MySQL command line

More on Joins

```
mysql> SELECT *
      FROM t1 INNER JOIN t2;

mysql> SELECT *
      FROM t1 INNER JOIN t2 ON t1.id = t2.id;
mysql> SELECT *
      FROM t1 JOIN t2 ON t1.id = t2.id;        -- default JOIN is INNER JOIN
mysql> SELECT *
      FROM t1 CROSS JOIN t2 ON t1.id = t2.id;  -- Also called CROSS JOIN

mysql> SELECT *
      FROM t1 INNER JOIN t2 WHERE t1.id = t2.id;  -- Use WHERE instead of ON
mysql> SELECT *
      FROM t1, t2 WHERE t1.id = t2.id;           -- Use "commas" operator to join
```

MySQL Queries – MySQL command line

More on Joins - Outer Join

- LEFT JOIN produces rows that are in the left table, but may not in the right table;
- RIGHT JOIN produces rows that are in the right table but may not in the left table.

```
mysql> SELECT *
       FROM t1 LEFT JOIN t2 ON t1.id = t2.id;

mysql> SELECT *
       FROM t1 LEFT JOIN t2 USING (id);

mysql> SELECT *
       FROM t1 RIGHT JOIN t2 ON t1.id = t2.id;

mysql> SELECT *
       FROM t1 RIGHT JOIN t2 USING (id);
```

MySQL Queries – MySQL command line

Sub-Query

```
mysql> SELECT suppliers.name from suppliers
       WHERE suppliers.supplierID
          NOT IN (SELECT DISTINCT supplierID from products_suppliers);

-- Supplier 'QQ Corp' now supplies 'Pencil 6B'
-- You need to put the SELECT subqueies in parentheses
mysql> INSERT INTO products_suppliers VALUES (
          (SELECT productID  FROM products  WHERE name = 'Pencil 6B'),
          (SELECT supplierID FROM suppliers WHERE name = 'QQ Corp'));

-- Supplier 'QQ Copr' no longer supplies any item
mysql> DELETE FROM products_suppliers
       WHERE supplierID = (SELECT supplierID FROM suppliers WHERE name = 'QQ Corp');
```

MySQL Queries – MySQL command line

Data and Time

```
-- Create a table 'patients' of a clinic
mysql> CREATE TABLE patients (
          patientID      INT UNSIGNED  NOT NULL AUTO_INCREMENT,
          name           VARCHAR(30)   NOT NULL DEFAULT '',
          dateOfBirth    DATE          NOT NULL,
          lastVisitDate  DATE          NOT NULL,
          nextVisitDate  DATE          NULL,
                      -- The 'Date' type contains a date value in 'yyyy-mm-dd'
          PRIMARY KEY (patientID)
       );

mysql> INSERT INTO patients VALUES
          (1001, 'Ah Teck', '1991-12-31', '2012-01-20', NULL),
          (NULL, 'Kumar', '2011-10-29', '2012-09-20', NULL),
          (NULL, 'Ali', '2011-01-30', CURDATE(), NULL);
-- Date must be written as 'yyyy-mm-dd'
-- Function CURDATE() returns today's date

-- Select patients who last visited on a particular range of date
mysql> SELECT * FROM patients
       WHERE lastVisitDate BETWEEN '2012-09-15' AND CURDATE()
       ORDER BY lastVisitDate;
```

MySQL Queries – MySQL command line

<u>Data and Time</u>

```
-- Create a table 'patients' of a clinic
mysql> CREATE TABLE patients (
          patientID      INT UNSIGNED  NOT NULL AUTO_INCREMENT,
          name           VARCHAR(30)   NOT NULL DEFAULT '',
          dateOfBirth    DATE          NOT NULL,
          lastVisitDate  DATE          NOT NULL,
          nextVisitDate  DATE          NULL,
                         -- The 'Date' type contains a date value in 'yyyy-mm-dd'
          PRIMARY KEY (patientID)
       );

mysql> INSERT INTO patients VALUES
          (1001, 'Ah Teck', '1991-12-31', '2012-01-20', NULL),
          (NULL, 'Kumar', '2011-10-29', '2012-09-20', NULL),
          (NULL, 'Ali', '2011-01-30', CURDATE(), NULL);
-- Date must be written as 'yyyy-mm-dd'
-- Function CURDATE() returns today's date

-- Select patients who last visited on a particular range of date
mysql> SELECT * FROM patients
       WHERE lastVisitDate BETWEEN '2012-09-15' AND CURDATE()
       ORDER BY lastVisitDate;
```

MySQL Queries – MySQL command line

Data and Time

```
-- Select patients who were born in a particular year and sort by birth-month
-- Function YEAR(date), MONTH(date), DAY(date) returns
--   the year, month, day part of the given date
mysql> SELECT * FROM patients
       WHERE YEAR(dateOfBirth) = 2011
       ORDER BY MONTH(dateOfBirth), DAY(dateOfBirth);

-- Select patients whose birthday is today
mysql> SELECT * FROM patients
       WHERE MONTH(dateOfBirth) = MONTH(CURDATE())
         AND DAY(dateOfBirth) = DAY(CURDATE());

-- List the age of patients
-- Function TIMESTAMPDIFF(unit, start, end) returns the difference in the unit specified
mysql> SELECT name, dateOfBirth, TIMESTAMPDIFF(YEAR, dateOfBirth, CURDATE()) AS age
       FROM patients
       ORDER BY age, dateOfBirth;
```

MySQL Queries – MySQL command line

Data and Time

```
-- List patients whose last visited more than 60 days ago
mysql> SELECT name, lastVisitDate FROM patients
       WHERE TIMESTAMPDIFF(DAY, lastVisitDate, CURDATE()) > 60;
-- Functions TO_DAYS(date) converts the date to days
mysql> SELECT name, lastVisitDate FROM patients
       WHERE TO_DAYS(CURDATE()) - TO_DAYS(lastVisitDate) > 60;

-- Select patients 18 years old or younger
-- Function DATE_SUB(date, INTERVAL x unit) returns the date
--   by subtracting the given date by x unit.
mysql> SELECT * FROM patients
       WHERE dateOfBirth > DATE_SUB(CURDATE(), INTERVAL 18 YEAR);
```

MySQL Queries – MySQL command line

More Date/Time Functions

```
mysql> SELECT YEAR(NOW()), MONTH(NOW()), DAY(NOW()), HOUR(NOW()), MINUTE(NOW()),
SECOND(NOW());

mysql> SELECT DAYNAME(NOW()), MONTHNAME(NOW()), DAYOFWEEK(NOW()),
DAYOFYEAR(NOW());

mysql> SELECT DATE_ADD('2012-01-31', INTERVAL 5 DAY);

mysql> SELECT DATE_SUB('2012-01-31', INTERVAL 2 MONTH);

mysql> SELECT DATE_FORMAT('2012-01-01', '%W %D %M %Y');
Sunday 1st January 2012
    -- %W: Weekday name
    -- %D: Day with suffix
    -- %M: Month name
    -- %Y: 4-digit year
    -- The format specifiers are case-sensitive
```

MySQL Queries – MySQL command line

Views

```
mysql> CREATE VIEW supplier_view
       AS
       SELECT suppliers.name as `Supplier Name`, products.name as `Product Name`
       FROM products
          JOIN suppliers ON products.productID = products_suppliers.productID
          JOIN products_suppliers ON suppliers.supplierID = products_suppliers.supplierID;

-- You can treat the VIEW defined like a normal table
mysql> SELECT * FROM supplier_view;

mysql> DROP VIEW IF EXISTS patient_view;

mysql> CREATE VIEW patient_view
       AS
       SELECT
          patientID AS ID,
          name AS Name,
          dateOfBirth AS DOB,
          TIMESTAMPDIFF(YEAR, dateOfBirth, NOW()) AS Age
       FROM patients
       ORDER BY Age, DOB;

mysql> SELECT * FROM patient_view WHERE Name LIKE 'A%';
```

MySQL Queries – MySQL command line

Transactions

```
mysql> CREATE TABLE accounts (
           name     VARCHAR(30),
           balance  DECIMAL(10,2)
       );

mysql> INSERT INTO accounts VALUES ('Paul', 1000), ('Peter', 2000);
mysql> SELECT * FROM accounts;

-- Transfer money from one account to another account
mysql> START TRANSACTION;
mysql> UPDATE accounts SET balance = balance - 100 WHERE name = 'Paul';
mysql> UPDATE accounts SET balance = balance + 100 WHERE name = 'Peter';
mysql> COMMIT;     -- Commit the transaction and end transaction
mysql> SELECT * FROM accounts;


mysql> START TRANSACTION;
mysql> UPDATE accounts SET balance = balance - 100 WHERE name = 'Paul';
mysql> UPDATE accounts SET balance = balance + 100 WHERE name = 'Peter';
mysql> ROLLBACK;    -- Discard all changes of this transaction and end Transaction
mysql> SELECT * FROM accounts;
```

MySQL Queries – MySQL command line

Transactions

```
-- Disable autocommit by setting it to false (0)
mysql> SET autocommit = 0;
mysql> UPDATE accounts SET balance = balance - 100 WHERE name = 'Paul';
mysql> UPDATE accounts SET balance = balance + 100 WHERE name = 'Peter';
mysql> COMMIT;
mysql> SELECT * FROM accounts;

mysql> UPDATE accounts SET balance = balance - 100 WHERE name = 'Paul';
mysql> UPDATE accounts SET balance = balance + 100 WHERE name = 'Peter';
mysql> ROLLBACK;
mysql> SELECT * FROM accounts;

mysql> SET autocommit = 1;   -- Enable autocommit
```

A transaction groups a set of operations into a unit that meets the ACID test:

- Atomicity: If all the operations succeed, changes are committed to the database. If any of the operations fails, the entire transaction is rolled back, and no change is made to the database. In other words, there is no partial update.
- Consistency: A transaction transform the database from one consistent state to another consistent state.
- Isolation: Changes to a transaction are not visible to another transaction until they are committed.
- Durability: Committed changes are durable and never lost.

MySQL Queries – MySQL command line

Backup

(For Windows)

```
-- Start a NEW "cmd"
> cd path-to-mysql-bin
> mysqldump -u root -p --databases southwind > "d:\myProject\backup_fs1030.sql"
```

(For Macs)

```
-- Start a NEW "terminal"
$ cd /usr/local/mysql/bin
$ ./mysqldump -u root -p --databases southwind > ~/Documents/backup_fs1030.sql
```

Restore

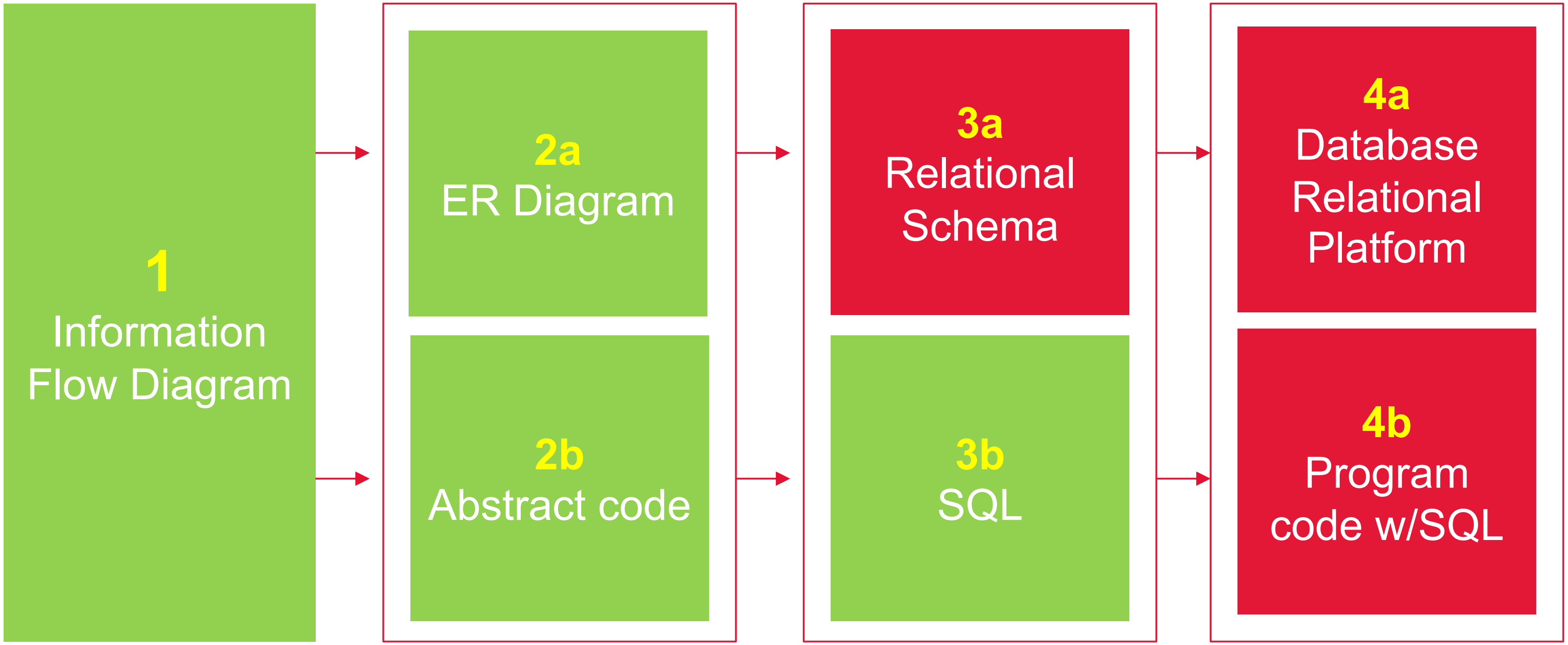
(For Windows)

```
-- Start a MySQL client
mysql> source d:/myProject/backup_fs1030.sql
   -- Provide absolute or relative filename of the script
   -- Use Unix-style forward slash (/) as path separator
```

(For Macs)

```
-- Start a MySQL client
mysql> source ~/Documents/backup_fs1030.sql
```

What we learned today



1
Information Flow Diagram

2a
ER Diagram

2b
Abstract code

3a
Relational Schema

3b
SQL

4a
Database Relational Platform

4b
Program code w/SQL

Analysis

Specification

Design

Implementation

Codds 12 Rule of RDBMS

Rule zero
This rule states that for a system to qualify as an **RDBMS**, it must be able to manage database entirely through the relational capabilities.

Rule 1: Information rule
All information(including metadata) is to be represented as stored data in cells of tables. The rows and columns have to be strictly unordered.

Rule 2: Guaranted Access
Each unique piece of data(atomic value) should be accesible by : **Table Name + Primary Key(Row) + Attribute(column)**.
NOTE: Ability to directly access via POINTER is a violation of this rule.

Rule 3: Systematic treatment of NULL
Null has several meanings, it can mean missing data, not applicable or no value. It should be handled consistently. Also, Primary key must not be null, ever. Expression on NULL must give null.

Codds 12 Rule of RDBMS

Rule 4: Active Online Catalog
Database dictionary(catalog) is the structure description of the complete **Database** and it must be stored online. The Catalog must be governed by same rules as rest of the database. The same query language should be used on catalog as used to query database.

Rule 5: Powerful and Well-Structured Language
One well structured language must be there to provide all manners of access to the data stored in the database. Example: **SQL**, etc. If the database allows access to the data without the use of this language, then that is a violation.

Rule 6: View Updation Rule
All the view that are theoretically updatable should be updatable by the system as well.

Rule 7: Relational Level Operation
There must be Insert, Delete, Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported.

Codds 12 Rule of RDBMS

Rule 8: Physical Data Independence
The physical storage of data should not matter to the system. If say, some file supporting table is renamed or moved from one disk to another, it should not effect the application.

Rule 9: Logical Data Independence
If there is change in the logical structure(table structures) of the database the user view of data should not change. Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.

Rule 10: Integrity Independence
The database should be able to enforce its own integrity rather than using other programs. Key and Check constraints, trigger etc, should be stored in Data Dictionary. This also make **RDBMS** independent of front-end.

Codds 12 Rule of RDBMS

Rule 11: Distribution Independence
A database should work properly regardless of its distribution across a network. Even if a database is geographically distributed, with data stored in pieces, the end user should get an impression that it is stored at the same place. This lays the foundation of **distributed database**.

Rule 12: Nonsubversion Rule
If low level access is allowed to a system it should not be able to subvert or bypass integrity rules to change the data. This can be achieved by some sort of looking or encryption.

SEED DATA

```
CREATE DATABASE ORG;
SHOW DATABASES;
USE ORG;

CREATE TABLE Worker (
   WORKER_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
   FIRST_NAME CHAR(25),
   LAST_NAME CHAR(25),
   SALARY INT(15),
   JOINING_DATE DATETIME,
   DEPARTMENT CHAR(25)
);

INSERT INTO Worker
   (WORKER_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT) VALUES
      (001, 'Monika', 'Arora', 100000, '14-02-20 09.00.00', 'HR'),
      (002, 'Niharika', 'Verma', 80000, '14-06-11 09.00.00', 'Admin'),
      (003, 'Vishal', 'Singhal', 300000, '14-02-20 09.00.00', 'HR'),
      (004, 'Amitabh', 'Singh', 500000, '14-02-20 09.00.00', 'Admin'),
      (005, 'Vivek', 'Bhati', 500000, '14-06-11 09.00.00', 'Admin'),
      (006, 'Vipul', 'Diwan', 200000, '14-06-11 09.00.00', 'Account'),
      (007, 'Satish', 'Kumar', 75000, '14-01-20 09.00.00', 'Account'),
      (008, 'Geetika', 'Chauhan', 90000, '14-04-11 09.00.00', 'Admin');
```

SEED DATA

```
CREATE TABLE Bonus (
   WORKER_REF_ID INT,
   BONUS_AMOUNT INT(10),
   BONUS_DATE DATETIME,
   FOREIGN KEY (WORKER_REF_ID)
      REFERENCES Worker(WORKER_ID)
ON DELETE CASCADE
);

INSERT INTO Bonus
   (WORKER_REF_ID, BONUS_AMOUNT, BONUS_DATE) VALUES
      (001, 5000, '16-02-20'),
      (002, 3000, '16-06-11'),
      (003, 4000, '16-02-20'),
      (001, 4500, '16-02-20'),
      (002, 3500, '16-06-11');

CREATE TABLE Title (
   WORKER_REF_ID INT,
   WORKER_TITLE CHAR(25),
   AFFECTED_FROM DATETIME,
   FOREIGN KEY (WORKER_REF_ID)
      REFERENCES Worker(WORKER_ID)
ON DELETE CASCADE
);
```

SEED DATA

```
INSERT INTO Title
    (WORKER_REF_ID, WORKER_TITLE, AFFECTED_FROM) VALUES
(001, 'Manager', '2016-02-20 00:00:00'),
(002, 'Executive', '2016-06-11 00:00:00'),
(008, 'Executive', '2016-06-11 00:00:00'),
(005, 'Manager', '2016-06-11 00:00:00'),
(004, 'Asst. Manager', '2016-06-11 00:00:00'),
(007, 'Executive', '2016-06-11 00:00:00'),
(006, 'Lead', '2016-06-11 00:00:00'),
(003, 'Lead', '2016-06-11 00:00:00');
```

MySQL Queries – Exercise

Import Seed Data

- Write An SQL Query To Fetch Unique Values Of DEPARTMENT From Worker Table.
- Write An SQL Query To Print All Worker Details From The Worker Table Order By FIRST_NAME Ascending.
- Write An SQL Query To Print Details For Workers With The First Name As “Vipul” And “Satish” From Worker Table.
- Write An SQL Query To Print Details Of The Workers Whose FIRST_NAME Contains ‘a’.
- Write An SQL Query To Print Details Of The Workers Whose SALARY Lies Between 100000 And 500000.
- Write An SQL Query To Print Details Of The Workers Who Are Also Managers.
- Write An SQL Query To Fetch Duplicate Records Having Matching Data In Some Fields Of A Table.
- Write An SQL Query To Fetch The List Of Employees With The Same Salary.
- Write An SQL Query To Fetch The Departments That Have Less Than Five People In It.
- Write An SQL Query To Print The Name Of Employees Having The Highest Salary In Each Department.

MySQL Queries – Exercise

Peter runs a small car rental company with 10 cars and 5 trucks. He engages you to design a web portal to put his operation online.

For the initial phase, the web portal shall provide these basic functions:

- Maintaining the records of the vehicles and customers.
- Inquiring about the availability of vehicle, and
- Reserving a vehicle for rental.

A customer record contains his/her name, address and phone number.

A vehicle, identified by the vehicle registration number, can be rented on a daily basis. The rental rate is different for different vehicles. There is a discount of 20% for rental of 7 days or more.

A customer can rental a vehicle from a start date to an end date. A special customer discount, ranging from 0-50%, can be given to preferred customers.

MySQL Queries – Exercise

```
DROP DATABASE IF EXISTS `rental_db`;
CREATE DATABASE `rental_db`;
USE `rental_db`;

-- Create `vehicles` table
DROP TABLE IF EXISTS `vehicles`;
CREATE TABLE `vehicles` (
   `veh_reg_no`  VARCHAR(8)    NOT NULL,
   `category`    ENUM('car', 'truck')  NOT NULL DEFAULT 'car',
                 -- Enumeration of one of the items in the list
   `brand`       VARCHAR(30)   NOT NULL DEFAULT '',
   `desc`        VARCHAR(256)  NOT NULL DEFAULT '',
                 -- desc is a keyword (for descending) and must be back-quoted
   `photo`       BLOB          NULL,   -- binary large object of up to 64KB
                 -- to be implemented later
   `daily_rate`  DECIMAL(6,2)  NOT NULL DEFAULT 9999.99,
                 -- set default to max value
   PRIMARY KEY (`veh_reg_no`),
   INDEX (`category`)  -- Build index on this column for fast search
) ENGINE=InnoDB;
   -- MySQL provides a few ENGINEs.
   -- The InnoDB Engine supports foreign keys and transactions
DESC `vehicles`;
SHOW CREATE TABLE `vehicles` \G
SHOW INDEX FROM `vehicles` \G

-- Create `customers` table
```

MySQL Queries – Exercise

```
 -- Create `customers` table
DROP TABLE IF EXISTS `customers`;
CREATE TABLE `customers` (
   `customer_id`  INT UNSIGNED  NOT NULL AUTO_INCREMENT,
                  -- Always use INT for AUTO_INCREMENT column to avoid run-over
   `name`         VARCHAR(30)   NOT NULL DEFAULT '',
   `address`      VARCHAR(80)   NOT NULL DEFAULT '',
   `phone`        VARCHAR(15)   NOT NULL DEFAULT '',
   `discount`     DOUBLE        NOT NULL DEFAULT 0.0,
   PRIMARY KEY (`customer_id`),
   UNIQUE INDEX (`phone`),  -- Build index on this unique-value column
   INDEX (`name`)           -- Build index on this column
) ENGINE=InnoDB;
DESC `customers`;
SHOW CREATE TABLE `customers` \G
SHOW INDEX FROM `customers` \G
```

MySQL Queries – Exercise

```
-- Create `rental_records` table
DROP TABLE IF EXISTS `rental_records`;
CREATE TABLE `rental_records` (
  `rental_id`    INT UNSIGNED  NOT NULL AUTO_INCREMENT,
  `veh_reg_no`   VARCHAR(8)    NOT NULL,
  `customer_id`  INT UNSIGNED  NOT NULL,
  `start_date`   DATE          NOT NULL DEFAULT '0000-00-00',
  `end_date`     DATE          NOT NULL DEFAULT '0000-00-00',
  `lastUpdated`  TIMESTAMP     NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    -- Keep the created and last updated timestamp for auditing and security
  PRIMARY KEY (`rental_id`),
  FOREIGN KEY (`customer_id`) REFERENCES `customers` (`customer_id`)
    ON DELETE RESTRICT ON UPDATE CASCADE,
    -- Disallow deletion of parent record if there are matching records here
    -- If parent record (customer_id) changes, update the matching records here
  FOREIGN KEY (`veh_reg_no`) REFERENCES `vehicles` (`veh_reg_no`)
    ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB;
DESC `rental_records`;
SHOW CREATE TABLE `rental_records` \G
SHOW INDEX FROM `rental_records` \G
```

MySQL Queries – Exercise

```
-- Inserting test records
INSERT INTO `vehicles` VALUES
   ('SBA1111A', 'car', 'NISSAN SUNNY 1.6L', '4 Door Saloon, Automatic', NULL, 99.99),
   ('SBB2222B', 'car', 'TOYOTA ALTIS 1.6L', '4 Door Saloon, Automatic', NULL, 99.99),
   ('SBC3333C', 'car', 'HONDA CIVIC 1.8L',  '4 Door Saloon, Automatic', NULL, 119.99),
   ('GA5555E', 'truck', 'NISSAN CABSTAR 3.0L',  'Lorry, Manual ', NULL, 89.99),
   ('GA6666F', 'truck', 'OPEL COMBO 1.6L',  'Van, Manual', NULL, 69.99);
   -- No photo yet, set to NULL
SELECT * FROM `vehicles`;

INSERT INTO `customers` VALUES
   (1001, 'Tan Ah Teck', '8 Happy Ave', '88888888', 0.1),
   (NULL, 'Mohammed Ali', '1 Kg Java', '99999999', 0.15),
   (NULL, 'Kumar', '5 Serangoon Road', '55555555', 0),
   (NULL, 'Kevin Jones', '2 Sunset boulevard', '22222222', 0.2);
SELECT * FROM `customers`;

INSERT INTO `rental_records` VALUES
  (NULL, 'SBA1111A', 1001, '2012-01-01', '2012-01-21', NULL),
  (NULL, 'SBA1111A', 1001, '2012-02-01', '2012-02-05', NULL),
  (NULL, 'GA5555E',  1003, '2012-01-05', '2012-01-31', NULL),
  (NULL, 'GA6666F',  1004, '2012-01-20', '2012-02-20', NULL);
SELECT * FROM `rental_records`;
```

MySQL Queries – Exercise

- Customer 'Tan Ah Teck' has rented 'SBA1111A' from today for 10 days. (Hint: You need to insert a rental record. Use a SELECT subquery to get the customer_id. Use CURDATE() (or NOW()) for today; and DATE_ADD(CURDATE(), INTERVAL x unit) to compute a future date.)

- List all rental records (start date, end date) with vehicle's registration number, brand, and customer name, sorted by vehicle's categories followed by start date.

- List all the expired rental records (end_date before CURDATE()).
- List the vehicles rented out on '2012-01-10' (not available for rental), in columns of vehicle registration no, customer name, start date and end date. (Hint: the given date is in between the start_date and end_date.)
- List all vehicles rented out today, in columns registration number, customer name, start date, end date.
- Similarly, list the vehicles rented out (not available for rental) for the period from '2012-01-03' to '2012-01-18'. (Hint: start_date is inside the range; or end_date is inside the range; or start_date is before the range and end_date is beyond the range.)
- List the vehicles (registration number, brand and description) available for rental (not rented out) on '2012-01-10' (Hint: You could use a subquery based on a earlier query).
- Similarly, list the vehicles available for rental for the period from '2012-01-03' to '2012-01-18'.
- Similarly, list the vehicles available for rental from today for 10 days.

- Foreign Key Test:
  - Try deleting a parent row with matching row(s) in child table(s), e.g., delete 'GA6666F' from vehicles table (ON DELETE RESTRICT).
  - Try updating a parent row with matching row(s) in child table(s), e.g., rename 'GA6666F' to 'GA9999F' in vehicles table. Check the effects on the child table rental_records (ON UPDATE CASCADE)

MySQL Queries – Exercise

- Foreign Key Test:
  - Try deleting a parent row with matching row(s) in child table(s), e.g., delete 'GA6666F' from vehicles table (ON DELETE RESTRICT).
  - Try updating a parent row with matching row(s) in child table(s), e.g., rename 'GA6666F' to 'GA9999F' in vehicles table. Check the effects on the child table rental_records (ON UPDATE CASCADE).
  - Remove 'GA6666F' from the database (Hints: Remove it from child table rental_records; then parent table vehicles.)

- Payments: A rental could be paid over a number of payments (e.g., deposit, installments, full payment). Each payment is for one rental. Create a new table called payments. Need to create columns to facilitate proper audit check (such as create_date, create_by, last_update_date, last_update_by, etc.)

- Staff: Keeping track of staff serving the customers. Create a new staff table. Assume that each transaction is handled by one staff, we can add a new column called staff_id in the rental_recordstable,

Thank you!

Submissions due before next class:

1. Take home assignment #1
2. Group Project Phase #1