a sql manual to

MINIONS

the database handbook

concept & content by

kamala kannan k suganya v

"kumbaya" ★★★★★ – Stuart

--NOT FOR PRINT--

Internal Circulation only

This material is prepared for educational purposes. Any attempt to duplicate this material is unethical.

Contents

Week 01

Story

The minions are happy creatures who like to serve the evil masters. In the course of earth's history, the minions have managed to overcome all crises to become one of the successful species on the planet.

They believe in the phrase "Survival of the evilest" and therefore are desperate to serve the evil master.

Each minion is always on the lookout for evil missions all over the globe.

Scenario

A minion has a name and can be of any gender. Average lifespan of a minion is ten years. They live in various countries and speak multiple languages. A minion can be hired by the master on hourly basis. The hiring charge of a minion is derived from its evilness factor.

Various missions are offered by evil masters, to spread evilness around the world. A mission can happen in any country within a specific duration (dd/mm/yyyy). Every mission has a predefined cost estimate and minion-hours. The evil masters hire the minions to complete the mission. A minion has to satisfy the pre-requisites to be eligible for the missions. Most evil minion is preferred normally to take part in a mission. And a mission can be of any of the following status: completed, underway, cancelled.

A master can offer one or more missions. Each master has a name, type and nationality.

Payment for a mission is done by the master. It can be paid in any currency. Currency conversion is done by the use of a common banana-currency (ƀ). Say if, ₹ 1 = ƀ 2.5, $ 1 = ƀ 5, then payment of ₹ 2 will be initiated to pay a bill of $ 1. Understand, a payment could be done only if a mission exists. Payment is always associated with date and amount.

There is a Minion Training Academy (MTA) which offers various courses for minions to develop their evilness. Senior minions handle the courses to the junior minions. Evilness can be increased by acquiring training at the academy. A minion is promoted to the next evil level after successful training.

Evilness is associated with a skill and a level factor (1-10)

ER Diagram

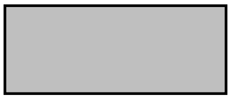
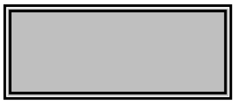
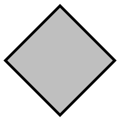
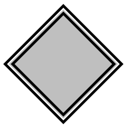
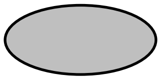
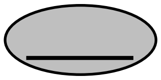
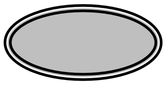
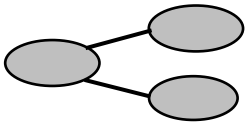
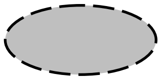
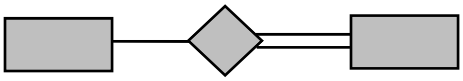
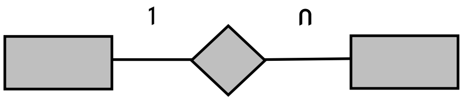
Proposed by Dr.Peter Chen, in 1970s

It is a conceptual model

The steps involved are as follows

🍌	Entity Identification	Identify the roles, events, locations, tangible things or concepts about which the end-users want to store data.
🍌	Identifying Relationship	Find the natural associations between pairs of entities using a relationship matrix.
🍌	Drawing a rough ERD	Put entities in rectangles and relationships in diagonals along the line segments connecting the entities.
🍌	Find Primary Key	Identify the data attribute(s) that uniquely identify one and only one occurrence of each entity.
🍌	Drawing a Key based ERD	Add the Primary key and its associated foreign key in other entities
🍌	Identify and Map attributes	Name the information details (fields) which are essential to the system under development.
🍌	Associate Cardinality	Determine the number of occurrences of one entity for a single occurrence of the related entity.
🍌	Drawing a fully attributed ERD	

Notation

Meaning	Symbol
Entity	
Weak Entity	
Relationship	
Identifying Relationship	
Attribute	
Key Attribute	
Multivalued Attribute	
Composite Attribute	
Derived Attribute	
Total Participation of E_2 in R	
Cardinality Ratio 1:N for E_1:E_2 in R	

WORK SHEET

ENTITY RELATIONSHIP MODEL

STEP 1 – ENTITY IDENTIFICATION

Note: Entity types fall into five classes: roles, events, locations, tangible things, or concepts

#	ENTITY_NAME	DESCRIPTION	TYPE
1.			
2.			
3.			
4.			
5.			

STEP 2 – RELATIONSHIP IDENTIFICATION

Note: Each row and column should have at least one relationship listed or else the entity associated with that row or column does not interact with the rest of the system.

ENTITY_1	ENTITY_1	ENTITY_2	ENTITY_3	ENTITY_4	ENTITY_5		
ENTITY_2							
ENTITY_3							
ENTITY_4							
ENTITY_5							

STEP 3 - ROUGH ERD

STEP 4 – MAPPING CARDINALITY

Note: 0 → No Instance, 1 → One Participating Instance, M → More Than One Participating Instance.

At each end of each connector joining rectangles, we need to place a symbol indicating the minimum and maximum number of instances of the adjacent rectangle there are for one instance of the rectangle at the other end of the relationship line.

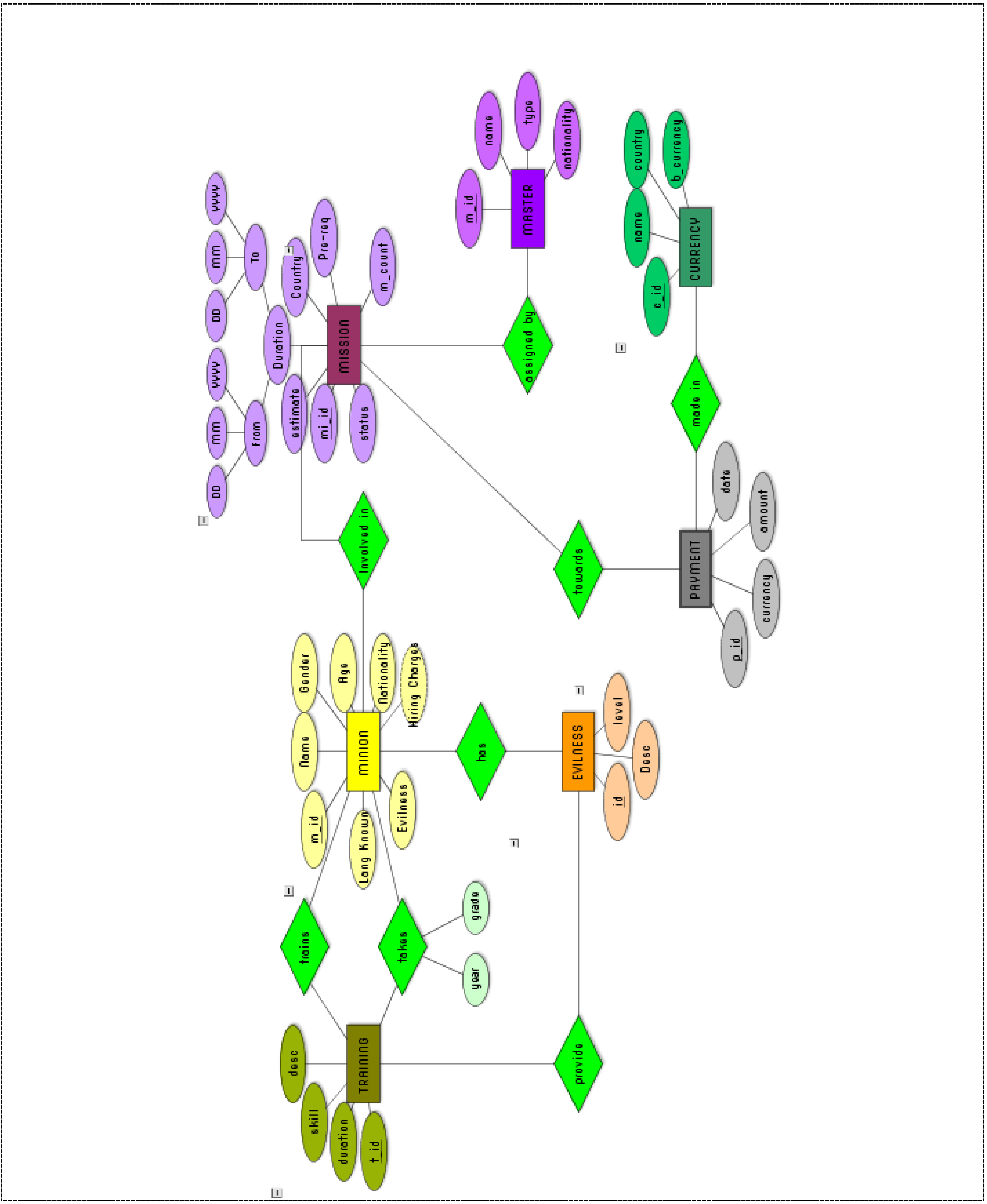
#	RELATIONSHIP	ENTITY_L → ENTITY_R	ENTITY_ R → ENTITY_L
1.			
2.			
3.			
4.			
5.			
6.			

STEP 5 – DEFINE PRIMARY KEY AND OTHER ATTRIBUTE

#	ENTITY	KEY	OTHER ATTRIBUTES
1.			
2.			
3.			
4.			
5.			

STEP 6 – FULLY ATTRIBUTED ERD

Note: Finally do a manual check whether the ER specify all the system data accurately.



Week 02

Translation of ER to Relational Schema

Entity Relationship Model is a graphical representation of entities and relationships, used to understand conceptually on how to organize the data within database or any other information systems. To store data, this conceptual model has to be translated to a relational schema. The translation process follows several principles with the intention to not lose any information.

Translation process is normally approximate as there exists less feasibility to capture all the conditions depicted in ER within the relational schema.

MAPPING ENTITY SETS

Create a table for the entity set.

Make each attribute of the entity set a field of the table, with an appropriate type.

Declare the field or fields comprising the primary key

```
Create table minion(
  m_id number(4),
  name varchar(15),
  gender char(1),
  age number(1),
  nationality varchar(25),
  hiring charge number(10,3),
  evilness number(2),
  lang_known varchar(30),
  primary key(m_id)
  );
```

MAPPING WEAK ENTITY SETS

Create a table for the weak entity set.

Make each attribute of the weak entity set a field of the table.

Add fields for the primary key attributes of the identifying owner.

Declare a foreign key constraint on these identifying owner fields.

Instruct the system to automatically delete any tuples in the table for which there are no owners.

```
Create table payment(
  mi_id number(4),
  p_id number(4),
  currency varchar(10),
  amount number(10,3),
  primary key(mi_id,p_id),
  foreign key(mi_id) references
  mission(mi_id)
  on delete cascade
  );
```

MAPPING OF BINARY 1:1 RELATIONSHIP

Method 1: Foreign Key Approach

Indentify two entities participating in the relation

Select the entity with total participation and add the primary key of the second entity as the foreign key.

Method 2: Merged Relation Approach

If the relationship is total, all the attributes of entities and the relationship is merged to form a single relation.

```
Create table mission(
  mi_id number(4),
  estimate number(10,3),
  m_count number(2),
  pre_req varchar(25),
  status varchar(15),
  country varchar(15),
  from_date date,
  to_date date,
  master(m_id) not null,
  primary key(mi_id),
  foreign key(m_id) references
  master(m_id)
  );
```

Method 3: Cross-Reference Approach

Create a table for the relationship set.

Add all primary keys of the participating entity sets as fields of the table.

Add a field for each attribute of the relationship, if it exists.

Declare a primary key using all key fields from the entity sets.

Declare foreign key constraints for all these fields from the entity sets.

```
Create table takes(
  m_id number(4),
  t_id number(4),
  year number(4),
  grade varchar(2),
  primary key(m_id,t_id),
  foreign key (m_id) references
  minion(m_id),
  foreign key (t_id) references
  training(t_id)
  );
```

MAPPING OF 1:N RELATIONSHIP

Identify the entity on the N-side of the relationship

Include the primary key of the 1-side entity as foreign key to the N-side entity

Add other simple attributes.

MAPPING OF M:N RELATIONSHIP

Create a table for the relationship set.

Add all primary keys of the participating entity sets as fields of the table.

Add a field for each attribute of the relationship.

Declare a primary key using the key fields from the source entity set only.

Declare foreign key constraints for all the fields from the source and target entity sets.

```
Create table assigns(
  m_id number(4),
  mi_id number(4),
  primary key(m_id),
  foreign key(m_id) references
  master(m_id),
  foreign key(mi_id) references
  mission(mi_id)
  );
```

Note: Because the assigned_by relation is many-to-one, we don't in fact need a whole table for the relation itself. However, this does slightly "pollute" the source entity table.

Alternate Method

Create a table for the source and target entity sets as usual.

Add every primary key field of the target as a field in the source.

Declare these fields as foreign keys.

```
Create table mission(
  mi_id number(4),
  estimate number(10,3),
  m_count number(2),
  pre_req varchar(25),
  status varchar(15),
  country varchar(15),
  from_date date,
  to_date date,
  master(m_id) not null,
  primary key(mi_id),
  foreign key(m_id) references
  master(m_id)
  );

Create table payment_made_in(
  p_id number(4),
  c_id number(4),
  p_date date,
  primary key(p_id,c_id,date),
  foreign key (p_id) references
  payment(p_id),
  foreign key (c_id) references
  currency(c_id)
  );
```

MAPPING OF MULTIVALUED ATTRIBUTES

create a new relation

Add the primary key of the entity to which the attribute belong to as an attribute and foreign key.

Declare the primary key to this relation as the combination of the attribute value and the entity's primary key.

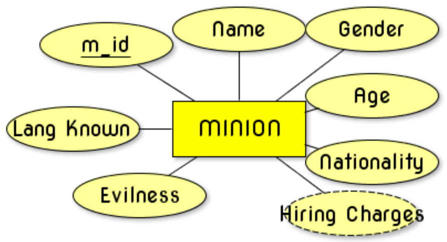
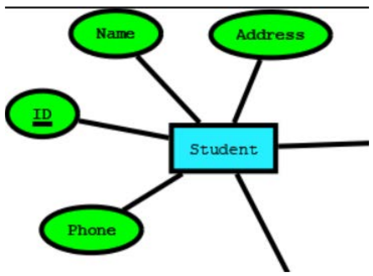
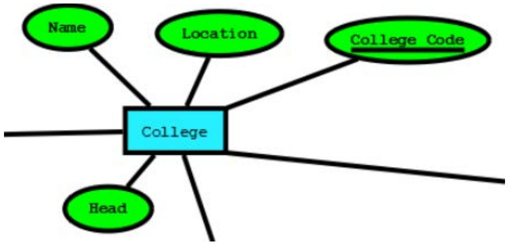
```
Create table languages_known(
  m_id number(4),
  language varchar(20),
  primary key(m_id,language),
  foreign key(m_id) references
  minion(m_id)
  );
```

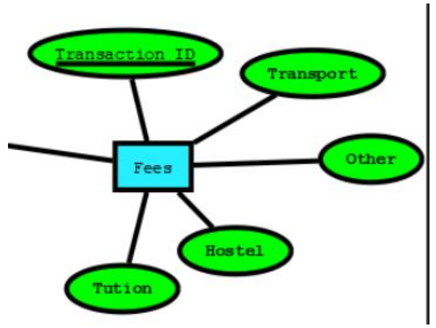
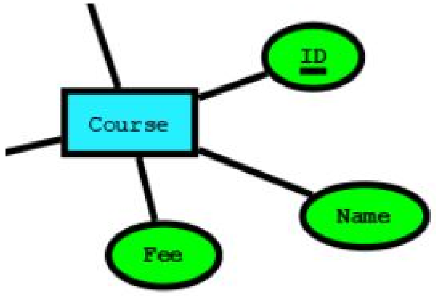
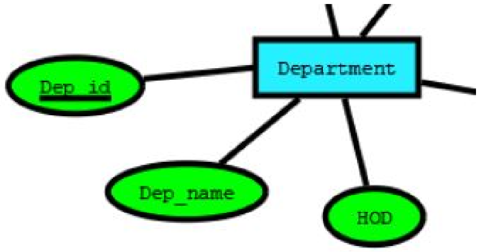
In summary, an/a

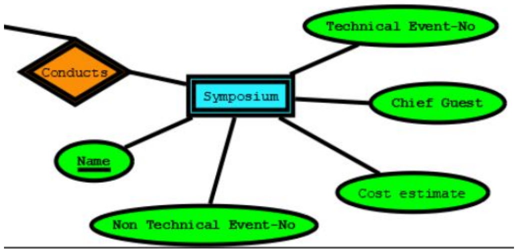
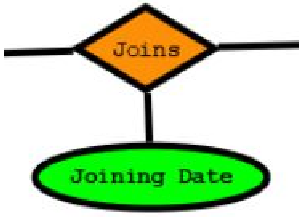
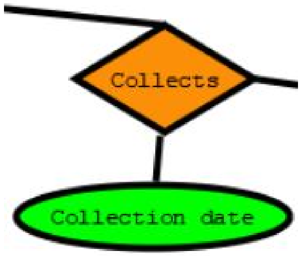
ENTITY TYPE	Translated as	RELATION
1:1 OR 1:N RELATIONSHIP		FOREIGN KEY (OR) RELATIO9NSHIP RELATION
M:N RELATIONSHIP		RELATON WITH TWO FOREIGN KEYS
SIMPLE ATTRIBUTE		ATTRIBUTE
COMPOSITE ATTRIBUTE		SET OF SIMPLE COMPONENT ATTRIBUTES
MULTI VALUED ATTRIBUTE		RELATION AND FOREIGN KEY
VALUE SET		DOMAIN
KEY ATTRIBUTE		PRIMARY KEY

WORK SHEET

ER-RELATIONAL MAPPING

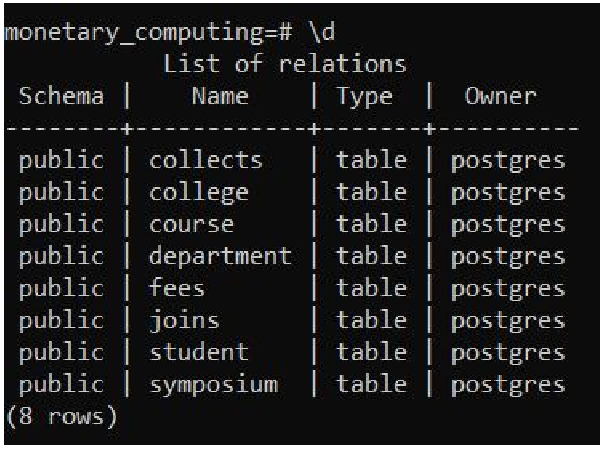
ER COMPONENT	TYPE/ PRIMARY KEY(S)	RELATED ENTITIES & THEIR PRIMARY KEY		CORRESPONDING RELATION SCHEMA
 m_id Name Gender Age Lang Known minion Nationality Evilness Hiring Charges	STRONG ENTITY			Create table minion(m_id number(4), name varchar(15), gender char(1), age number(1), nationality varchar(25), hiring charge number(10,3), evilness number(2), lang_known varchar(30));
	m_ID			
	Strong Entity	Department	dep_id	create table student(stu_id int, stu_name varchar(30), address varchar(50), phone int, dep_id int, primary key(stu_id), foreign key(dep_id) references department(dep_id));
	stu_id			
	Strong Entity			create table college(clg_code int, clg_name varchar(30), location varchar(50), head varchar(30), primary key(clg_code));
	clg_code			

ER COMPONENT	TYPE/ PRIMARY KEY(S)	RELATED ENTITIES & THEIR PRIMARY KEY		CORRESPONDING RELATION SCHEMA
 Transaction ID Transport Other Fees Hostel Tution	Strong Entity t_id			create table fees(t_id int, tution_fee int, hostel_fee int, transport_fee int, other int, primary key(t_id));
	Strong Entity course_id	Department	dep_id	create table course(course_id int, course_name varchar(20), course_fee int, dep_id int, primary key(course_id), foreign key(dep_id) references department(dep_id));
	Strong Entity dep_id			create table department(dep_id int, dep_name varchar(15), hod varchar(30), primary key(dep_id));

ER COMPONENT	TYPE/ PRIMARY KEY(S)	RELATED ENTITIES & THEIR PRIMARY KEY		CORRESPONDING RELATION SCHEMA
 Technical Event-No Conducts Symposium Chief Guest Name Cost estimate Non Technical Event-No	Weak Entity	Department	dep_id	create table symposium(dep_id int, symp_name varchar(20), chief_guest varchar(30), tech_event_no int, non_tech_event_no int, cost int, primary key(dep_id,symp_name), foreign key(dep_id) references department(dep_id) on delete cascade);
	name,dep_id			
	Relation			create table joins(stu_id int, clg_code int, join_date date, primary key(stu_id,clg_code), foreign key(stu_id) references student(stu_id), foreign key(clg_code) references college(clg_code));
	stu_id, clg_code	Student	stu_id	
		College	clg_id	
	Relation	College	clg_id	create table collects(col_date date, t_id int, clg_code int, primary key(t_id,clg_code), foreign key(t_id) references fees(t_id), foreign key(clg_code) references college(clg_code));
	clg_code, t_id	Fees	t_id	

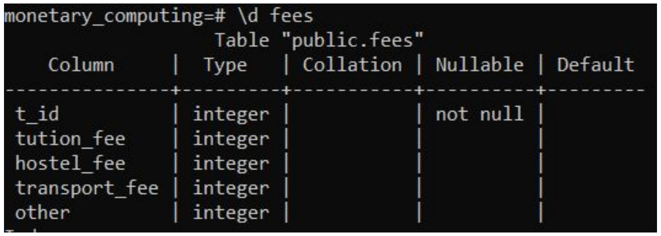
ER COMPONENT	TYPE/ PRIMARY KEY(S)	RELATED ENTITIES & THEIR PRIMARY KEY		CORRESPONDING RELATION SCHEMA

MINION DATABASE SCHEMA



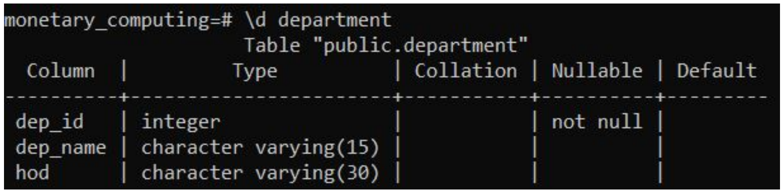
```
monetary_computing=# \d
           List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | collects   | table | postgres
 public | college    | table | postgres
 public | course     | table | postgres
 public | department | table | postgres
 public | fees       | table | postgres
 public | joins      | table | postgres
 public | student    | table | postgres
 public | symposium  | table | postgres
(8 rows)
```

List of relations



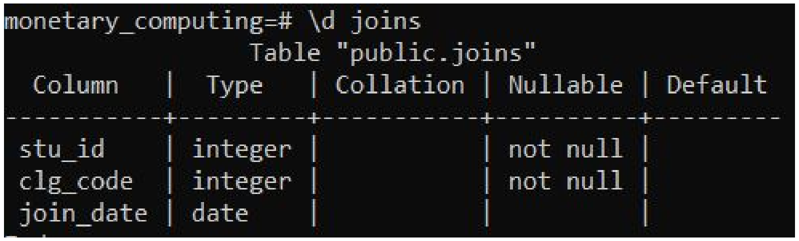
```
monetary_computing=# \d fees
                 Table "public.fees"
    Column     |  Type   | Collation | Nullable | Default
---------------+---------+-----------+----------+---------
 t_id          | integer |           | not null |
 tution_fee    | integer |           |          |
 hostel_fee    | integer |           |          |
 transport_fee | integer |           |          |
 other         | integer |           |          |
```

Entity: Fees



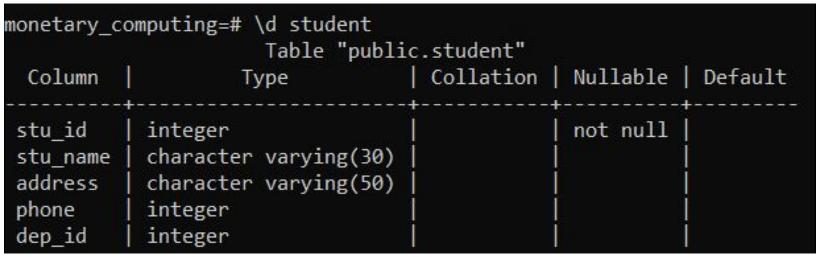
```
monetary_computing=# \d department
                  Table "public.department"
  Column  |         Type          | Collation | Nullable | Default
----------+-----------------------+-----------+----------+---------
 dep_id   | integer               |           | not null |
 dep_name | character varying(15) |           |          |
 hod      | character varying(30) |           |          |
```

Entity: Department



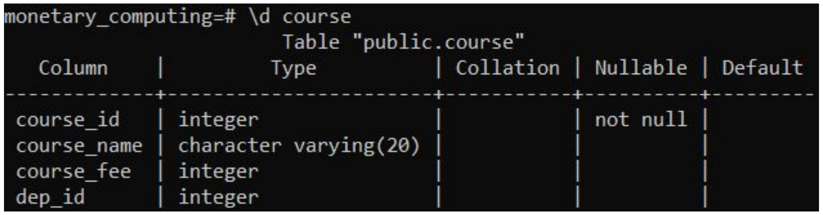
```
monetary_computing=# \d joins
                 Table "public.joins"
  Column   |  Type   | Collation | Nullable | Default
-----------+---------+-----------+----------+---------
 stu_id    | integer |           | not null |
 clg_code  | integer |           | not null |
 join_date | date    |           |          |
```

Relation: Joins



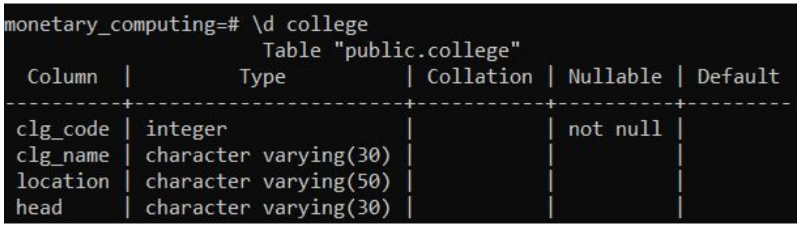
```
monetary_computing=# \d student
                   Table "public.student"
  Column  |         Type          | Collation | Nullable | Default
----------+-----------------------+-----------+----------+---------
 stu_id   | integer               |           | not null |
 stu_name | character varying(30) |           |          |
 address  | character varying(50) |           |          |
 phone    | integer               |           |          |
 dep_id   | integer               |           |          |
```

Entity: Student



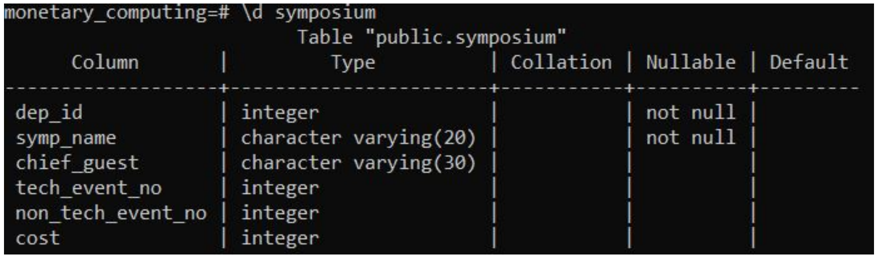
```
monetary_computing=# \d course
                     Table "public.course"
   Column    |         Type          | Collation | Nullable | Default
-------------+-----------------------+-----------+----------+---------
 course_id   | integer               |           | not null |
 course_name | character varying(20) |           |          |
 course_fee  | integer               |           |          |
 dep_id      | integer               |           |          |
```

Entity: Course



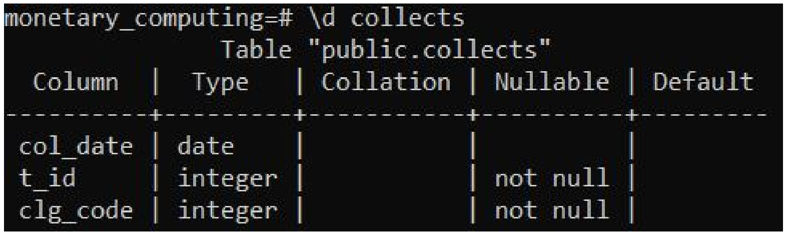
```
monetary_computing=# \d college
                   Table "public.college"
  Column  |         Type          | Collation | Nullable | Default
----------+-----------------------+-----------+----------+---------
 clg_code | integer               |           | not null |
 clg_name | character varying(30) |           |          |
 location | character varying(50) |           |          |
 head     | character varying(30) |           |          |
```

Entity: College



```
monetary_computing=# \d symposium
                       Table "public.symposium"
      Column       |         Type          | Collation | Nullable | Default
-------------------+-----------------------+-----------+----------+---------
 dep_id            | integer               |           | not null |
 symp_name         | character varying(20) |           | not null |
 chief_guest       | character varying(30) |           |          |
 tech_event_no     | integer               |           |          |
 non_tech_event_no | integer               |           |          |
 cost              | integer               |           |          |
```

Entity: Symposium



```
monetary_computing=# \d collects
              Table "public.collects"
  Column  |  Type   | Collation | Nullable | Default
----------+---------+-----------+----------+---------
 col_date | date    |           |          |
 t_id     | integer |           | not null |
 clg_code | integer |           | not null |
```

Relation: Collects

Introduction to SQL

SQL – Structured Query Language

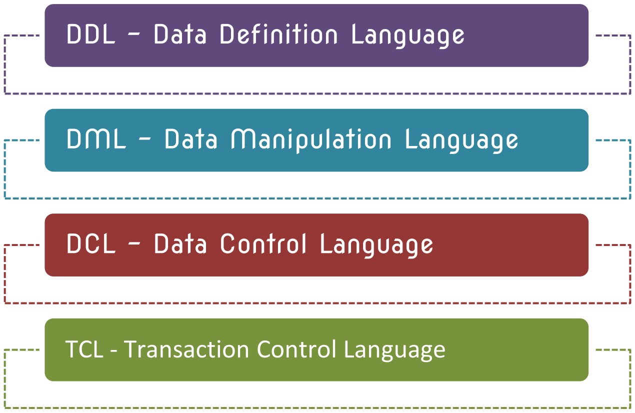
SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

SQL Process:

On executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc.

SQL Commands:



DDL – Data Definition Language

DML – Data Manipulation Language

DCL – Data Control Language

TCL - Transaction Control Language

What is RDBMS?

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

Eg : MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

What is table?

The data in RDBMS is stored in database objects called tables. The table is a collection of related data entries and it consists of columns and rows.

What is field?

Every table is broken up into smaller entities called fields.

A field is a column in a table that is designed to maintain specific information about every record in the table.

What is record or row?

A record, also called a row of data, is each individual entry that exists in a table.

What is NULL value?

A NULL value in a table is a field with no value. { NULL ≠ 0 }

SQL Constraints:

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Column level constraints are applied only to one column whereas table level constraints are applied to the whole table.

CONSTRAINT	DESCRIPTION
NOT NULL	Ensures that a column cannot have NULL value
DEFAULT	Provides a default value for a column when none is specified.
UNIQUE	Ensures that all values in a column are different.
PRIMARY	Uniquely identified each rows/records in a database table.
FOREIGN	Uniquely identified a rows/records in any another database table.
CHECK	Ensures that all values in a column satisfy certain conditions.
INDEX	Use to create and retrieve data from the database very quickly.

Data Integrity:

The following categories of the data integrity exist with each RDBMS:

Entity Integrity: There are no duplicate rows in a table.

Domain Integrity: Enforces valid entries for a given column by restricting the type, the format, or the range of values.

Referential integrity: Rows cannot be deleted, which are used by other records.

User-Defined Integrity: Enforces some specific business rules that do not fall into entity, domain or referential integrity.

COMMAND CREATE TABLE

PURPOSE To create a table, the basic structure to hold user data, specifying this information:

column definitions

integrity constraints

the table's tablespace

storage characteristics

data from an arbitrary query

SYNTAX

```
CREATE TABLE table_name(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
    ...
  column_n datatype [ NULL | NOT NULL ],
  CONSTRAINT constraint_name PRIMARY KEY (column1,
  column2, ... column_n),
  CONSTRAINT fk_column FOREIGN KEY (column1, column2,
  ... column_n)REFERENCES parent_table (column1,
  column2, ... column_n)ON DELETE CASCADE,
  CONSTRAINT constraint_name CHECK (column_name
  condition) [DISABLE],
  CONSTRAINT constraint_name UNIQUE (uc_col1, uc_col2,
  ... uc_col_n)
  );
```

MEANING

schema : is the schema containing the table. If you omit schema, Oracle assumes the table is in your own schema.

table : is the name of the table to be created.

column : specifies the name of a column of the table. The number of columns in a table can range from 1 to 254.

datatype : is the datatype of a column.

DEFAULT : specifies a value to be assigned to the column if a subsequent INSERT statement omits a value for the column. The datatype of the expression must match the datatype of the column. A DEFAULT expression cannot contain references to other columns, the pseudocolumns CURRVAL, NEXTVAL, LEVEL, and ROWNUM, or date constants that are not fully specified.

column_constraint : defines an integrity constraint as part of the column definition.

table_constraint : defines an integrity constraint as part of the table definition.

COMMAND ALTER TABLE

PURPOSE To alter the definition of a table in one of these ways:

to add a column

to add an integrity constraint

to redefine a column (datatype, size, default value)

to modify storage characteristics or other parameters

to enable, disable, or drop an integrity constraint or trigger

SYNTAX

```
ALTER TABLE <Table_Name>
        [ADD {    { column datatype [DEFAULT expr]
[column_constraint] ...
                  | table_constraint}
             |   ( { column datatype [DEFAULT expr]
[column_constraint] ...
                  | table_constraint}
               [, { column datatype [DEFAULT expr]
[column_constraint] ...
                  | table_constraint} ] ... ) } ]
        [MODIFY {   column [datatype] [DEFAULT expr]
[column_constraint] ...
                |  (column [datatype] [DEFAULT expr]
[column_constraint] ...
     [, column datatype [DEFAULT expr] [column_constraint]
...] ...) } ]
        [DROP drop_clause] ...
```

MEANING

ADD : adds a column or integrity constraint.

MODIFY : modifies a the definition of an existing column. If you omit any of the optional parts of the column definition (datatype, default value, or column constraint), these parts remain unchanged.

DEFAULT : specifies a default value for a new column or a new default for an existing column. Oracle assigns this value to the column if a subsequent INSERT statement omits a value for the column. The datatype of the default value must match the datatype specified for the column. A DEFAULT expression cannot contain references to other columns, the pseudocolumns CURRVAL, NEXTVAL, LEVEL, and ROWNUM, or date constants that are not fully specified.

column_constraint : adds or removes a NOT NULL constraint to or from and existing column.

table_constraint : adds an integrity constraint to the table.

COMMAND

DROP TABLE

PURPOSE

To remove a table and all its data from the database.

SYNTAX

```
DROP TABLE [schema.]table

        [CASCADE CONSTRAINTS]
```

MEANING

schema : is the schema containing the table. If you omit schema, Oracle assumes the table is in your own schema.

table : is the name of the table to be dropped.

CASCADE CONSTRAINTS : drops all referential integrity constraints that refer to primary and unique keys in the dropped table. If you omit this option, and such referential integrity constraints exist, Oracle returns an error and does not drop the table.

COMMAND	COMMIT
PURPOSE	To end your current transaction and make permanent all changes performed in the transaction. This command also erases all savepoints in the transaction and releases the transaction's locks. You can also use this command to manually commit an in-doubt distributed transaction.
SYNTAX	COMMIT [WORK] [COMMENT 'text' | FORCE 'text' [, integer]]
MEANING	WORK : is supported only for compliance with standard SQL. The statements COMMIT and COMMIT WORK are equivalent. COMMENT : specifies a comment to be associated with the current transaction. The 'text' is a quoted literal of up to 50 characters that Oracle stores in the data dictionary view DBA_2PC_PENDING along with the transaction ID if the transaction becomes in-doubt. FORCE : manually commits an in-doubt distributed transaction.

COMMAND	ROLLBACK
PURPOSE	To undo work done in the current transaction. You can also use this command to manually undo the work done by an in-doubt distributed transaction.
SYNTAX	ROLLBACK [WORK] [TO [SAVEPOINT] savepoint | FORCE 'text']
MEANING	WORK : is optional and is provided for ANSI compatibility. TO : rolls back the current transaction to the specified savepoint. If you omit this clause, the ROLLBACK statement rolls back the entire transaction

COMMAND SELECT

PURPOSE To retrieve data from one or more tables, views, or snapshots.

SYNTAX

```
SELECT [DISTINCT | ALL] { *
                             | { [schema.]{table | view |
snapshot}.*
                               | expr }  [ [AS] c_alias ]
                            [, { [schema.]{table | view |
snapshot}.*
                               | expr } [ [AS] c_alias ]  ]
... }
         FROM [schema.]{table | view | subquery |
snapshot}[@dblink] [t_alias]
           [, [schema.]... ] ...
         [WHERE condition ]
         [ [START WITH condition] CONNECT BY condition]
         [GROUP BY expr [, expr] ... [HAVING condition] ]
         [{UNION | UNION ALL | INTERSECT | MINUS} SELECT
command ]
         [ORDER BY {expr|position} [ASC | DESC]
                [, {expr|position} [ASC | DESC]] ...]
         [FOR UPDATE [OF [[schema.]{table | view}.]column
                      [, [[schema.]{table | view}.]column]
...] [NOWAIT] ]
```

MEANING DISTINCT : returns only one copy of each set of duplicate rows selected. Duplicate rows are those with matching values for each expression in the select list.

ALL : returns all rows selected, including all copies of duplicates. The default is ALL.

* : selects all columns from all tables, views, or snapshots listed in the FROM clause.

table.* | view.* | snapshot.* : selects all columns from the specified table, view, or snapshot. You can use the schema qualifier to select from a table, view, or snapshot in a schema other than your own. If you are using Trusted Oracle, the * does not select the ROWLABEL column. To select this column, you must explicitly specify it in the select list.

expr : selects an expression, usually based on columns values, from one of the tables, views, or snapshots in the FROM clause. A column name in this list can only contain be qualified with schema if the table, view, or snapshot containing the column is qualified with schema in the FROM clause.

c_alias : provides a different name for the column expression and causes the alias to be used in the column heading. A column alias does not affect the actual name of the column. Column aliases can be referenced in the ORDER BY clause but in no other clauses in a statement.

table | view | subquery | snapshot : is the name of a table, view, or snapshot from which data is selected. A subquery is treated in the same fashion as a view.

dblink : is complete or partial name for a database link to a remote database where the table, view, or snapshot is located. Note that this database need not be an Oracle7 database. If you omit dblink, Oracle assumes that the table, view, or snapshot is on the local database.

t_alias : provides a different name for the table, view, or snapshot for the purpose of evaluating the query and is most often used in a correlated query. Other references to the table, view, or snapshot throughout the query must refer to the alias.

WHERE : restricts the rows selected to those for which the condition is TRUE. If you omit this clause, Oracle returns all rows from the tables, views, or snapshots in the FROM clause.

START WITH | CONNECT BY : returns rows in a hierarchical order.

GROUP BY : groups the selected rows based on the value of expr for each row and returns a single row of summary information for each group.

HAVING : restricts the groups of rows returned to those groups for which the specified condition is TRUE. If you omit this clause, Oracle returns summary rows for all groups.

UNION | UNION ALL | INTERSECT | MINUS : combines the rows returned by two SELECT statement using a set operation.

AS : can optionally precede a column alias. To comply with the ANSI SQL92 standard, column aliases must be preceded by the AS keyword.

ORDER BY : orders rows returned by the statement.

> expr - orders rows based on their value for expr. The expression is based on columns in the select list or columns in the tables, views, or snapshots in the FROM clause.
>
> position - orders rows based on their value for the expression in this position of the select list.
>
> ASC | DESC - specifies either ascending or descending order. ASC is the default.

The ORDER BY clause can reference column aliases defined in the SELECT list.

FOR UPDATE : locks the selected rows.

NOWAIT : returns control to you if the SELECT statement attempts to lock a row that is locked by another user. If you omit this clause, Oracle waits until the row is available and then returns the results of the SELECT statement.

EXERCISE

1. Identify the relations for the Minion Database.
2. Analyze the primary key and their dependencies in other relations.
3. Create/alter/drop identified relations using SQL commands.

DATASHEET

Minions (19+1)

- Stuart
- Bob
- Kevin
- Dave
- Mark
- Phil
- Liza
- Mike
- Paul
- Lance
- Zugi
- Steve
- .
- .
- .
- .
- .
- .

Master (9+1)

- Adolf Hitler
- Gru
- Frankenstein
- Ivan Dracula
- Osama
- Nero
- Mojo Jojo
- Loki
- Lex Luthor
- Megatron
- .

Currency (5)

- INR
- USD
- EUR
- GBP
- JPY

Language (10)

- English
- French
- Spanish
- Chinese
- Urdu
- German
- .
- .
- .
- .

Country (10)

- USA
- Germany
- Japan
- France
- India
- .
- .
- .
- .
- .

Skills (10)

- Sneeze
- Itch
- Ticklish
- Yawn
- .
- .
- .
- .
- .

Mission: Hide pages hereafter

Minion Hired: Kevin

Evil Master: Zugi

Duration: Until next week

You have successfully completed Week 02. Come back next week!!