

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	Topic	Page
01	Overview, ER (Case Study)	4
02	ER-Relational Mapping, EER	11
03	DDL, DML	29
04	Simple SQL Queries, Joins, Grouping, Ordering	37
05	Functions - Strings, Numeric, Date/Time	47
06	Set Operators	54
07	Sub queries	61
08	Views	68
09	PL/SQL - Block	73
10	PL/SQL - Cursors	76
11	PL/SQL - Loops and Controls	86
12	PL/SQL - Procedures	92
13	PL/SQL - Functions	95
14	PL/SQL - Triggers	98
15	Project Deliverables	

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 vilest" 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(₹). Say if, ₹1 = ₹ 2.5, \$ 1 = ₹ 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



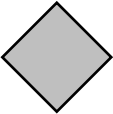
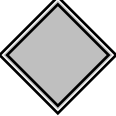



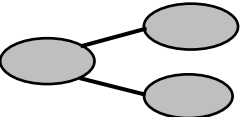

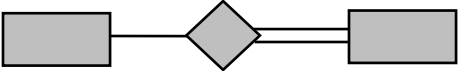
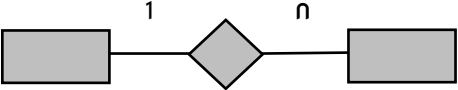
Proposed by Dr.Peter Chen, in 1970s

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	

ENTITY RELATIONSHIP MODEL

WORK SHEET

STEP 1 - ENTITY IDENTIFICATION

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

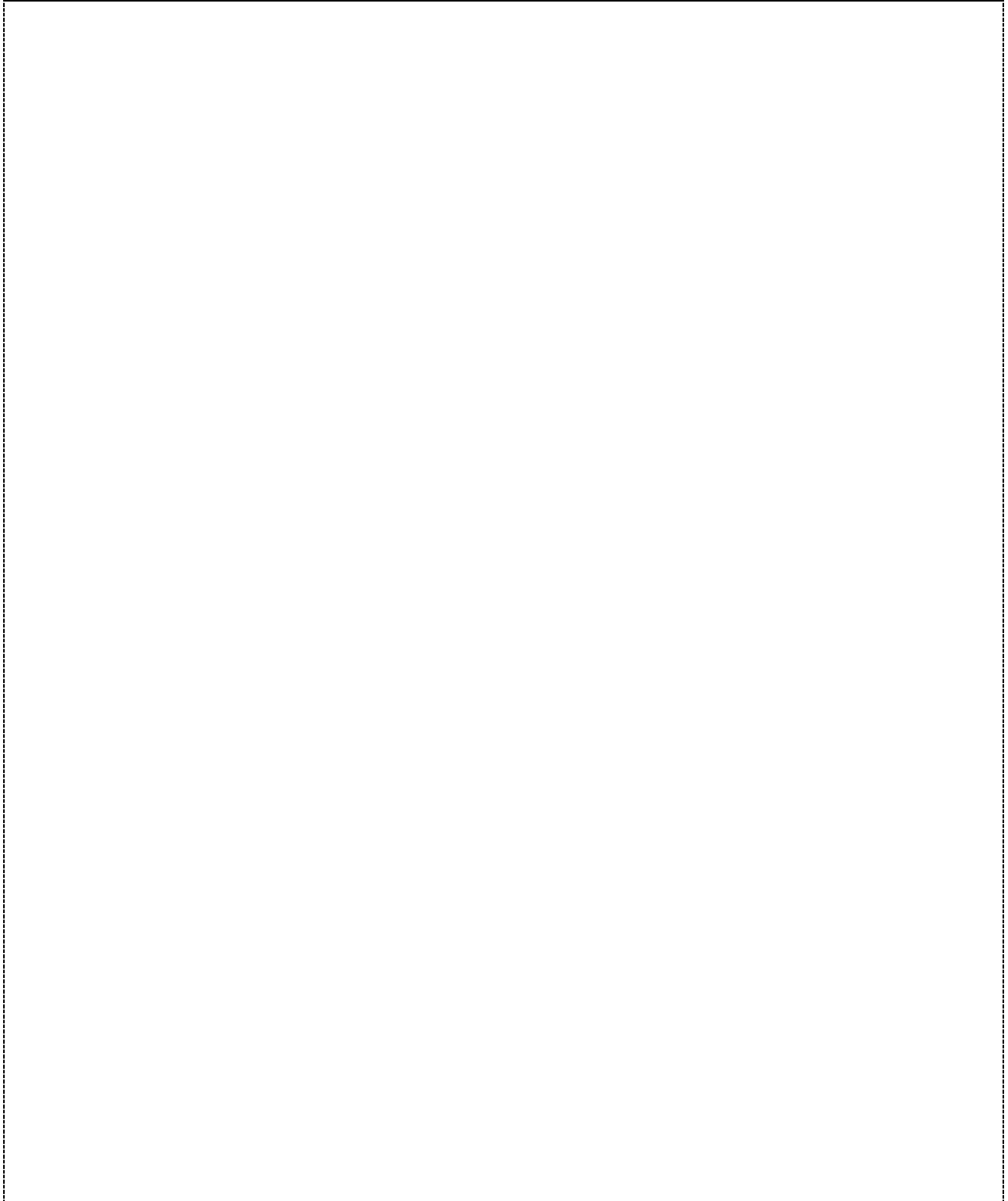
#	ENTITY_NAME	DESCRIPTION	TYPE
1.			
2.			
3.			
4.			
5.			

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.

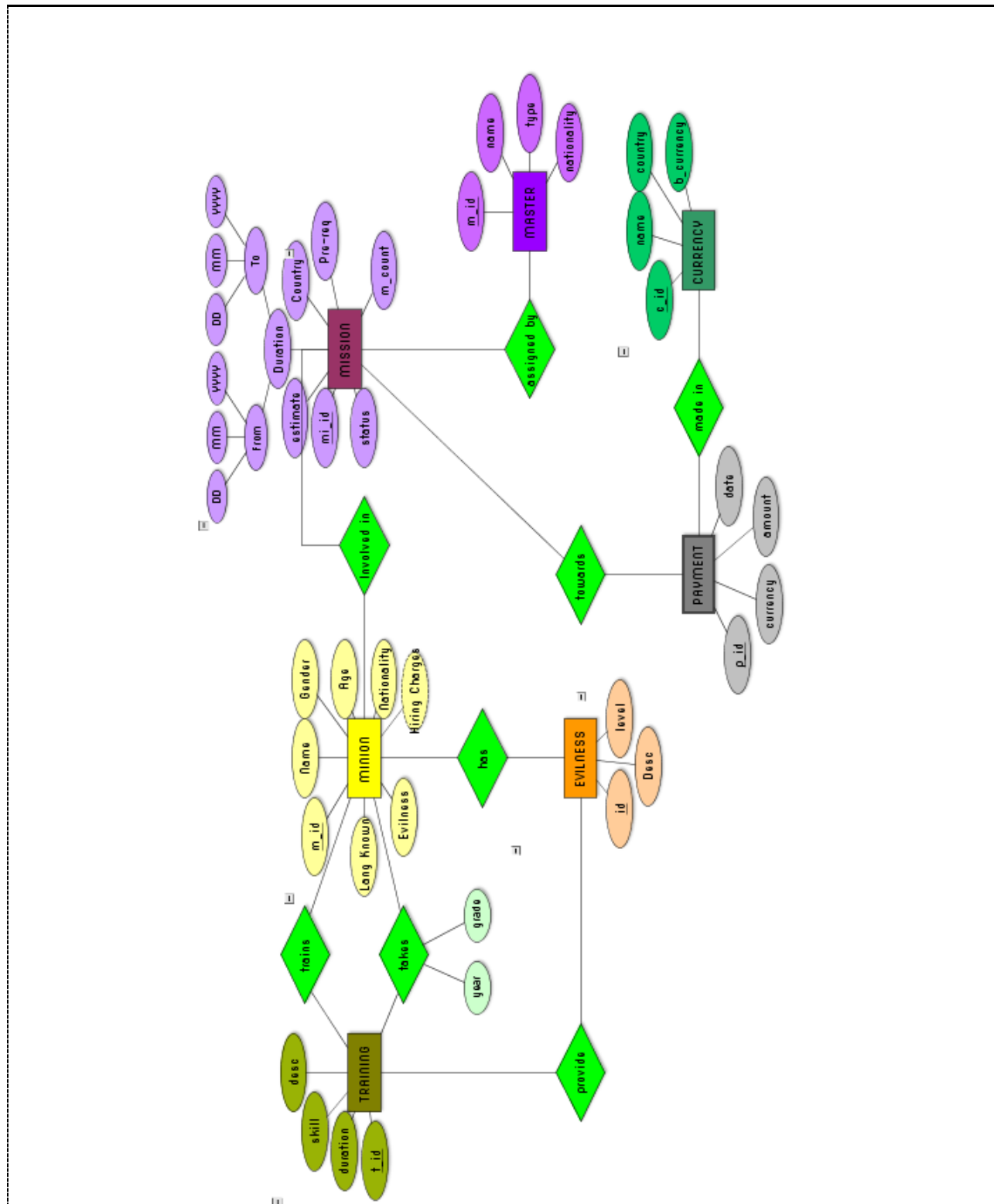
#	RELATIONSHIP	ENTITY_L → ENTITY_R	ENTITY_R → ENTITY_L
1.			
2.			
3.			
4.			
5.			
6.			

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

Identify 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.

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 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 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_byrelation 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.

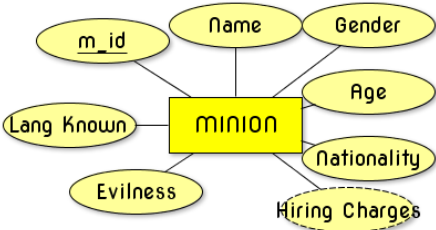
Declare the primary key to this relation as the combination of the attribute value and the entity's primary key.

```
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) RELATIONSHIP RELATION
M:N RELATIONSHIP		RELATION 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

ER-RELATIONAL MAPPING

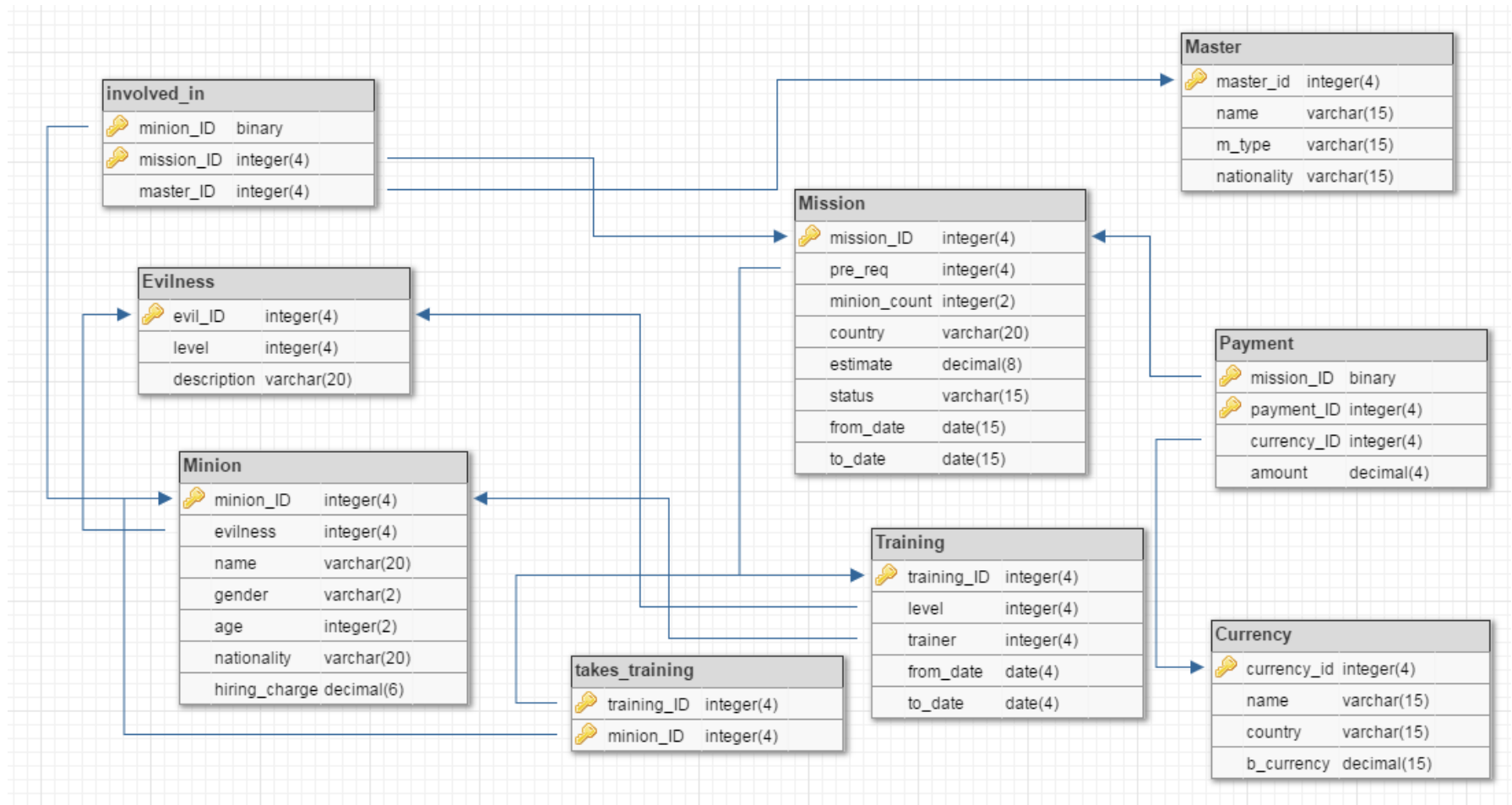
ER COMPONENT	TYPE/ PRIMARY KEY(S)	RELATED ENTITIES & THEIR PRIMARY KEY		CORRESPONDING RELATION SCHEMA
	STRONG ENTITY			<pre>Create table minion(m_id number(4), name varchar(15), gender char(1), age number(1), nationality varchar(25), hiring charge number(10,3), evilness number(2), lang_known varchar(30));</pre>
	m_ID			

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

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

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

MINION DATABASE SCHEMA



Introduction to SQL

SQL – Structured Query Language

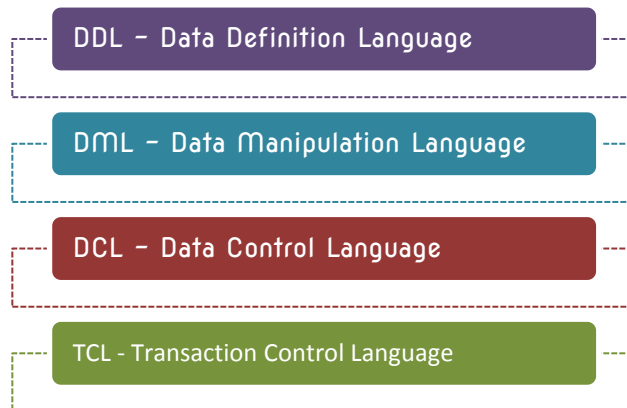
SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as standard database language.

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:



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 \neq 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	<p>To create a table, the basic structure to hold user data, specifying this information:</p> <ul style="list-style-type: none">column definitionsintegrity constraintsthe table's tablespacestorage characteristicsdata from an arbitrary query
SYNTAX	<pre>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));</pre>

MEANING	<p>schema : is the schema containing the table. If you omit schema, Oracle assumes the table is in your own schema.</p> <p>table : is the name of the table to be created.</p> <p>column : specifies the name of a column of the table. The number of columns in a table can range from 1 to 254.</p> <p>datatype : is the datatype of a column.</p> <p>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.</p> <p>column_constraint : defines an integrity constraint as part of the column definition.</p> <p>table_constraint : defines an integrity constraint as part of the table definition.</p>
---------	--

COMMAND	ALTER TABLE
PURPOSE	<p>To alter the definition of a table in one of these ways:</p> <ul style="list-style-type: none"> 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	<pre> 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] ... </pre>

MEANING	<p>ADD : adds a column or integrity constraint.</p> <p>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.</p> <p>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.</p> <p>column_constraint : adds or removes a NOT NULL constraint to or from an existing column.</p> <p>table_constraint : adds an integrity constraint to the table.</p>
---------	---

COMMAND	DROP TABLE
PURPOSE	To remove a table and all its data from the database.
SYNTAX	<p>DROP TABLE [schema.]table</p> <p>[CASCADE CONSTRAINTS]</p>
MEANING	<p>schema : is the schema containing the table. If you omit schema, Oracle assumes the table is in your own schema.</p> <p>table : is the name of the table to be dropped.</p> <p>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.</p>

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	<p>WORK : is supported only for compliance with standard SQL. The statements COMMIT and COMMIT WORK are equivalent.</p> <p>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.</p> <p>FORCE : manually commits an in-doubt distributed transaction.</p>

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

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.

Week 03

Data Definition Language

The Data Definition Language (DDL) manages table and index structure. The most basic items of DDL are the CREATE, ALTER, RENAME and DROP statements:








CREATE creates an object (a table, for example) in the database.

DROP deletes an object in the database, usually irretrievably.

ALTER modifies the structure an existing object in various ways—for example, adding a column to an existing table.

Best practice:

When creating tables you should consider following these guidelines:

-  Tables: Use upper case and singular form in table names – not plural, e.g., “STUDENT” (not students)
-  Columns: Use Pascal notation, e.g., “StudentId” or “Student_Id”
-  Primary Key: If the table name is “COURSE”, name the Primary Key column “CourseId”, etc. “Always” use Integer for Primary Keys.
-  Use UNIQUE constraint for other columns that needs to be unique, e.g. RoomNumber
-  Specify Required Columns (NOT NULL) – i.e., which columns that need to have data or not
-  Standardize on few/these Data Types: int, float, varchar(x), datetime, bit
-  Avoid abbreviations! (Use RoomNumber – not RoomNo, RoomNr, ...)

Exercise

For the relations identified during the ER to relational mapping, create tables using DDL commands in SQL* PLUS.

Note:

1. Understand the dependencies across relations and create tables accordingly.
2. Analyze the necessity of constraints before table creation
3. Work with all possible constraints like- NOT NULL, UNIQUE, CHECK, DEFAULT, PRIMARY KEY and FOREIGN KEY
4. Experiment the constraint enforcement using 'CREATE TABLE' and 'ALTER TABLE'.

CODE SHEET

```
CREATE TABLE minion
(
    minion_id number(4) NOT NULL,
    name varchar(10),
    gender varchar(1),
    age number(2)CHECK (AGE >= 18 ),
    nationality varchar(15),
    hiring_charge number(5,3),
    evilness number(4),
    CONSTRAINT minion_pkey PRIMARY KEY (minion_id),
    CONSTRAINT minion_evilness_fkey FOREIGN KEY (evilness)
        REFERENCES public.evilness (evil_id)
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
);

ALTER TABLE minion MODIFY SALARY hiring_charge (6,3) DEFAULT 500.00;

ALTER TABLE minion ALTER COLUMN hiring_charge DROP DEFAULT;

ALTER TABLE <table_name> MODIFY <attr_name> INT NOT NULL UNIQUE;

ALTER TABLE <table_name> ADD CONSTRAINT myUniqueConstraint
UNIQUE(attr1, attr2);

ALTER TABLE <table_name> DROP CONSTRAINT myUniqueConstraint;

ALTER TABLE minion MODIFY AGE INT NOT NULL CHECK (AGE >= 18 );

ALTER TABLE CUSTOMERS ADD CONSTRAINT myCheckConstraint CHECK(AGE >=
18);

ALTER TABLE minion ADD CONSTRAINT PK_minion PRIMARY KEY (minion_id);

ALTER TABLE minion DROP PRIMARY KEY ;

ALTER TABLE involved_in ADD FOREIGN KEY (minion_id) REFERENCES
minion(minion_id);

ALTER TABLE minion DROP FOREIGN KEY;
```

Data Manipulation Language

The Data Manipulation Language (DML) is the subset of SQL used to add, update and delete data. The acronym CRUD refers to all of the major functions that need to be implemented in a relational database application to consider it complete. Each letter in the acronym can be mapped to a standard SQL statement:

Syntax: INSERT

Single Row Insert:	<code>INSERT INTO table_name VALUES (value1, value2, value3,....);</code>
Single Row/ Specific Column:	<code>INSERT INTO table_name (column1, column2, column3,....) VALUES (value1, value2, value3,.....);</code>
Multiple Row from another table:	<code>INSERT INTO table-name (column1, column2,) SELECT column1, column2, FROM another_table_name</code>
Specific Rows from another table:	<code>INSERT INTO table-name (column1, column2,) SELECT column1, column2, FROM another_table_name WHERE column = '';</code>
Interactive Multi Row Insert:	<code>Insert into table-name values('&column1','&column2');</code>

Syntax: UPDATE

Single Column Update:	<code>UPDATE table-name set column-name = value where condition;</code>
Multi Column Update:	<code>UPDATE table-name set column1 = value, column2 = value where condition;</code>

Syntax: DELETE

All Column delete:	<code>DELETE from table-name;</code>
Specific Column deletion:	<code>DELETE from table-name where condition;</code>

Exercise

1. Insert the values as given below for all the tables. Use the table.sql file to create the required schema. Experiment the various syntax's in INSERT TABLE command.

Master			
Master_id	name	mttype	nationality
101	Adolf Hitler	person	Germany
102	Gru	cartoon	USA
103	Frankenstein	fictional	England
104	Ivan Dracula	fictional	Russia
105	Osama	person	Saudi Arabia
106	Nero	cartoon	Europe
107	Mojo Jojo	cartoon	USA
108	Loki	fictional	USA
109	Lex Luthor	fictional	USA
110	Megatron	cartoon	Cybertron

Evilness		
evil_id	level	description
201	1	Sneeze
202	2	Itch
203	3	Tickle
204	4	Yawn

Minion						
minion_id	evilness	name	gender	age	nationality	hiring_charge
301	203	Stuart	M	6	USA	100
302	204	Bob	M	8	Japan	140
303	202	Kevin	M	4	India	75
304	204	Dave	M	9	Russia	120
305	202	Mark	M	5	Germany	80
306	201	Phil	M	10	USA	90
307	204	Liza	F	3	Japan	150
308	203	Mike	M	6	India	90
309	202	Paul	M	9	Russia	85
310	204	Lance	F	6	England	70
311	202	Zugi	F	9	India	90
312	203	Steve	M	7	USA	110

Training				
training_ID	level	trainer	from_date	to_date
401	4	304	05-JAN-2017	15-JAN-2017
402	3	312	07-JAN-2017	10-JAN-2017
403	2	302	09-JAN-2017	24-JAN-2017
404	1	308	11-JAN-2017	15-JAN-2017

Mission							
mission_id	pre_req	m_count	country	estimate	status	from_date	to_date
501	3	1	Japan	1200	underway	14-JAN-17	18-JAN-17
502	4	2	USA	2500	underway	16-JAN-17	22-JAN-17
503	3	1	England	2200	underway	02-FEB-17	10-FEB-17
504	2	3	India	2400	completed	18-DEC-16	27-DEC-16
505	4	4	China	4000	cancelled	-	-

Involved_in		
minion_id	mission_id	master_id
302	501	110
301	502	107
306	502	107
310	503	103
303	504	104
308	504	104
311	504	104

Payment				
mission_id	payment_id	currency_id	amount	date
504	7303	601	765	27-DEC-2016
504	7308	601	765	27-DEC-2016
504	7311	601	637.5	27-DEC-2016

Takes_training	
training_id	minion_id
402	303
401	301
403	306
402	311

currency			
currency_id	name	country	b_currency
601	INR	India	4
602	Japanese YEN	Japan	8
603	Euro	Germany	4.5
604	USD	USA	7
605	Pound Sterling	England	9.5
606	Ruble	Russia	12
607	Riyal	Saudi Arabia	10

2. Insert a tuple to currency table, to depict the currency of Europe as Euro and the corresponding b_currency as 15.
3. Commit the data
4. Update the nationality of Dave as Japan.
5. Create a Savepoint A
6. Modify the completion date of the training which is offered to acquire level 4 in evilness.
7. Create a Savepoint B
8. As the minion count is not sufficient to trigger Frankenstein 's mission, revise the status as cancelled
9. Create a new table "Country" with attributes country_code and country_name. populate two tuples of your choice.
10. Create a Savepoint C
11. Delete some rows of the table "Country" using appropriate conditions.
12. Roll back to C. Write the inference.
13. Roll back to B. Write the inference w.r.t the Country table.

CODE SHEET

```
INSERT INTO master(
    master_id, name, mtype, nationality)
VALUES (101, 'Adolf Hitler', 'person', 'Germany');

INSERT ALL
    INTO master VALUES (101, 'Adolf Hitler', 'person', 'Germany')
    INTO master VALUES (102, 'Gru', 'cartoon', 'USA')
select * from dual;

INSERT INTO master VALUES(&master_id,&name','&mtype','&nationality');

CREATE TABLE demo_tbl(
    no NUMBER(3),
    name VARCHAR2(50)
);

INSERT INTO demo_tbl (no, name)
    SELECT no, name FROM
    users_info;

UPDATE master SET NATIONALITY = 'Europe'

UPDATE master SET NATIONALITY = 'Europe' WHERE master_id = 102

DELETE FROM master

DELETE FROM master WHERE master_id >200;
```

Week 04

Simple SQL Queries

SQL SELECT statement is used to query or retrieve data from a table in the database. A query may retrieve information from specified columns or from all of the columns in the table.

Syntax

```
SELECT column_list FROM table-name  
[WHERE Clause]  
[GROUP BY clause]  
[HAVING clause]  
[ORDER BY clause];
```

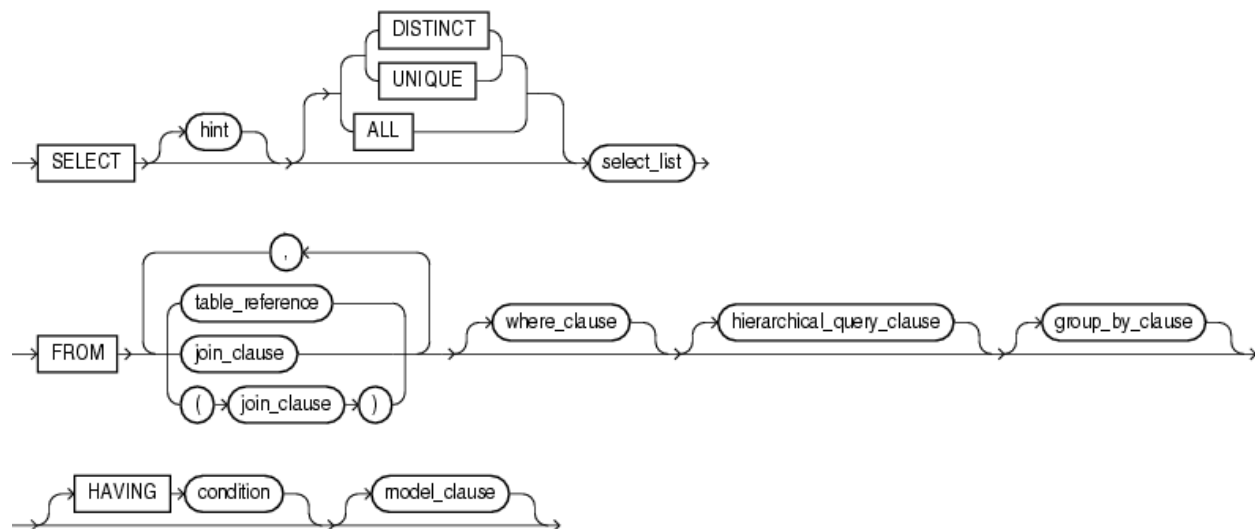


Figure 4.1: Select Statement Options

Select Clause:

The SELECT clause is mandatory. It specifies a list of columns to be retrieved from the tables in the FROM clause.

```
SELECT [ALL|DISTINCT] select-list
```

From Clause:

The 'FROM clause' always follows the SELECT clause. It lists the tables accessed by the query.

FROM S [,R]

Where Clause:

The 'WHERE clause' filters rows from the FROM clause tables. 'Where clause' has a logical predicate which returns SQL logical value -- true, false or unknown. The comparison can be done with a literal or any values of another column.

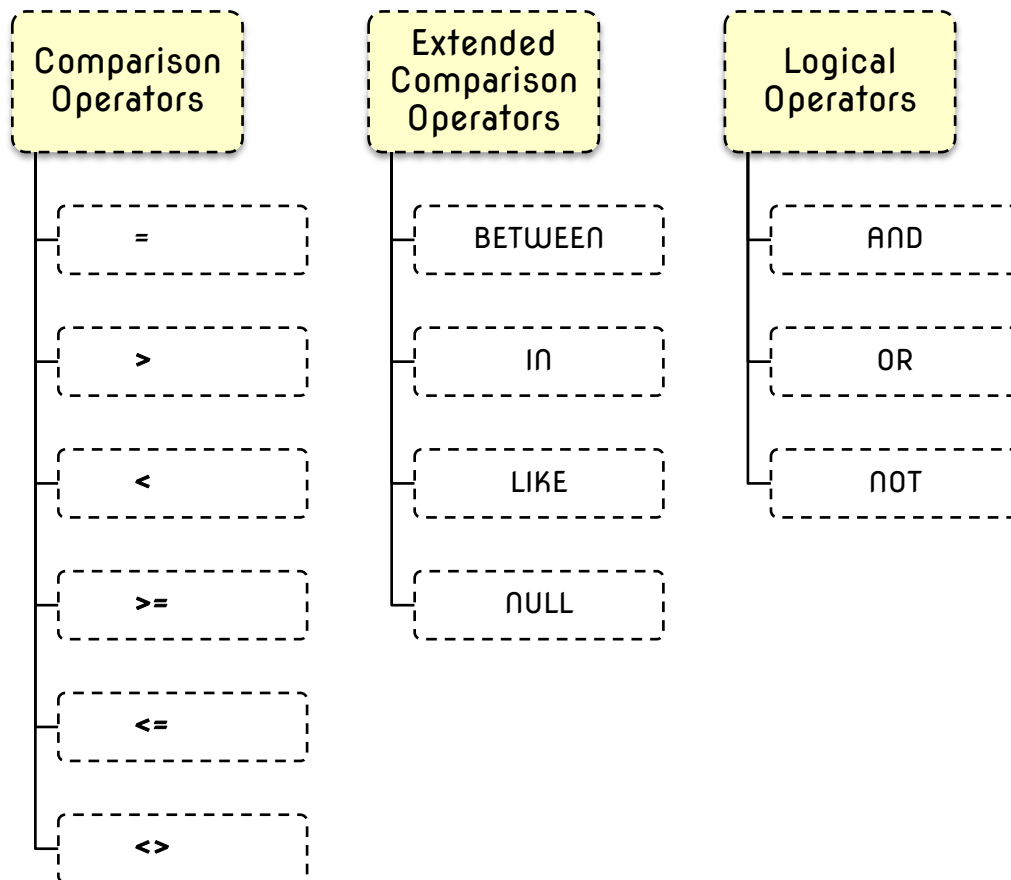


Figure 4.2: Operators

Joins

Joins are used to query data from more than one relation. Tuples from one relation can be joined with the tuples of another relation on satisfying the condition(s).

If the attributes (column name) of the relations are ambiguous, qualify with table prefixes aka Aliases. Prefixes are optional for the unique attributes (that are part of join operation) but, use of prefixes increases the overall performance.

Joins uses AND operator to specify the necessary constraints. The Inner join is also known as equi join will retrieve tuples if all the conditions (where, one among them would be an equality based constraint) mentioned are satisfied. Theta join retrieve tuples from the participating relation that satisfy the conditions apart from equality. Both Inner join and Theta join retrieves the tuples which satisfy the constraints; but, Outer join also include the tuples even if there exist no matching across the other relation.

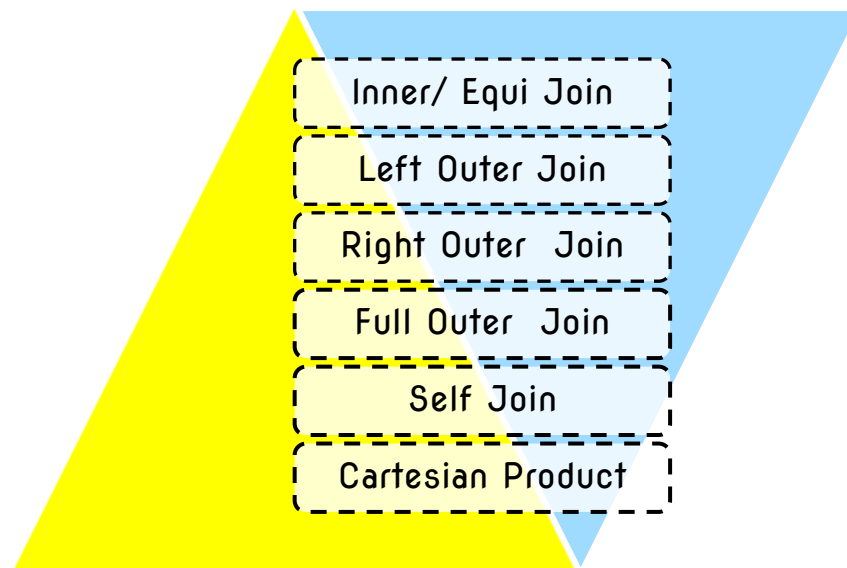
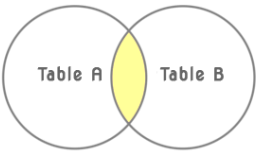
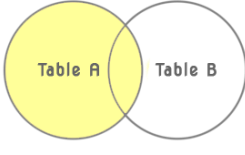
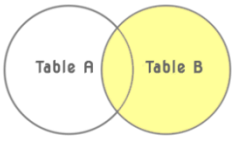
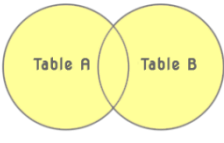
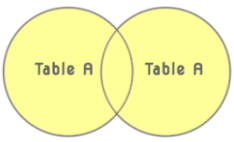


Figure 4.3: Types of Joins

The left outer join returns a result table with the matched data of two tables then remaining rows of the left table and null for the right table's column.

The right outer join returns a result table with the matched data of two tables then remaining rows of the right table and null for the left table's columns.

The full outer join returns a result table with the matched data of two table then remaining rows of both left table and then the right table.

<p>INNER JOIN</p> 	<p>Select all records from Table A and Table B, where the join condition is met.</p>	<p>LEFT JOIN</p> 	<p>Select all records from Table A, along with records from Table B for which the join condition is met (if at all).</p>
<p>RIGHT JOIN</p> 	<p>Select all records from Table B, along with records from Table A for which the join condition is met (if at all).</p>	<p>FULL JOIN</p> 	<p>Select all records from Table A and Table B, regardless of whether the join condition is met or not</p>
<p>SELF JOIN</p> 	<p>A table joined with itself.</p>		

Analysis

To understand the difference between the Cartesian product and Join, Consider the following tables:

MINION(minion_id, **evilness**, name, gender, age, nationality, hiring_charge) – 12 tuples

EVILNESS(**evil_id**, level, description) – 4 tuples

Cartesian product will provide the result of 48 tuples (all combinations across 12 tuples of minion and 4 tuples of evilness)

But, Join (in specific inner join) combines the tuples based on a condition. If the condition is satisfied, the rows are retrieved. But understand, Inner join internally executes as condition based selection on a Cartesian product.

The query,

```
select * from minion, evilness
```

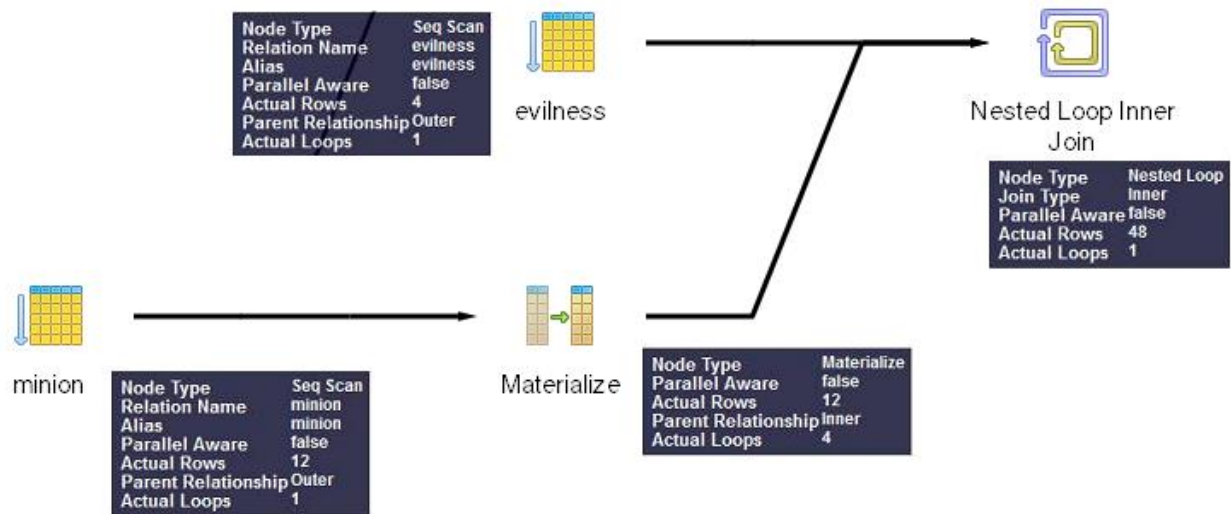


Figure 4.4: Cartesian Product: Minion and Evilness

The query,

```
select *
from minion m, evilness e
where m.evilness = e.evil_id
```

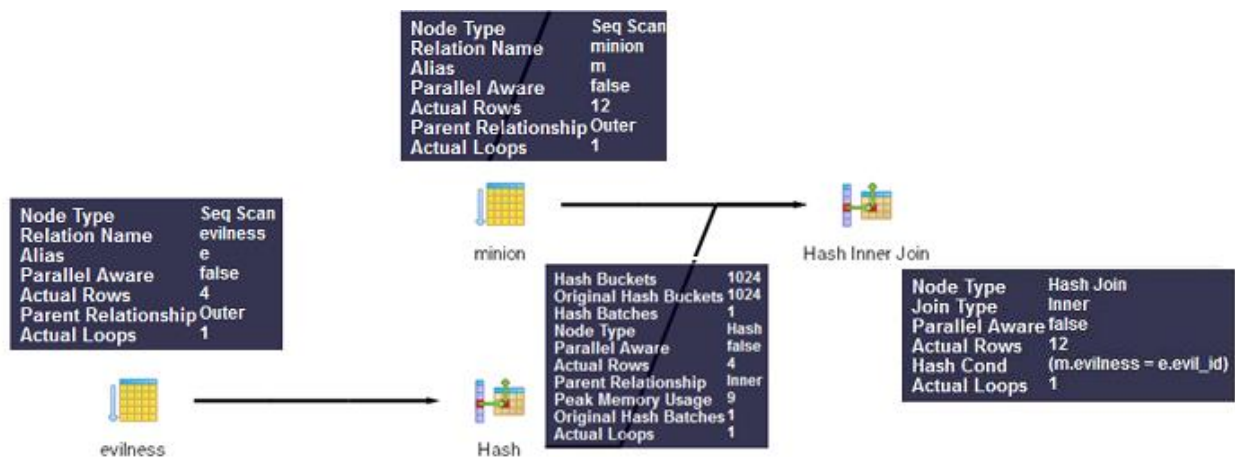


Figure 4.5: Join of Minion and Evilness

Aggregation

SQL Aggregate functions return a single value, using values in a table column which is of numeric datatype.

FUNCTION	DESCRIPTION
MIN	returns the smallest value in a given column
MAX	returns the largest value in a given column
SUM	returns the sum of the numeric values in a given column
AVG	returns the average value of a given column
COUNT	returns the total number of values in a given column
COUNT(*)	returns the number of rows in a table

Table 4.1: Aggregate Functions

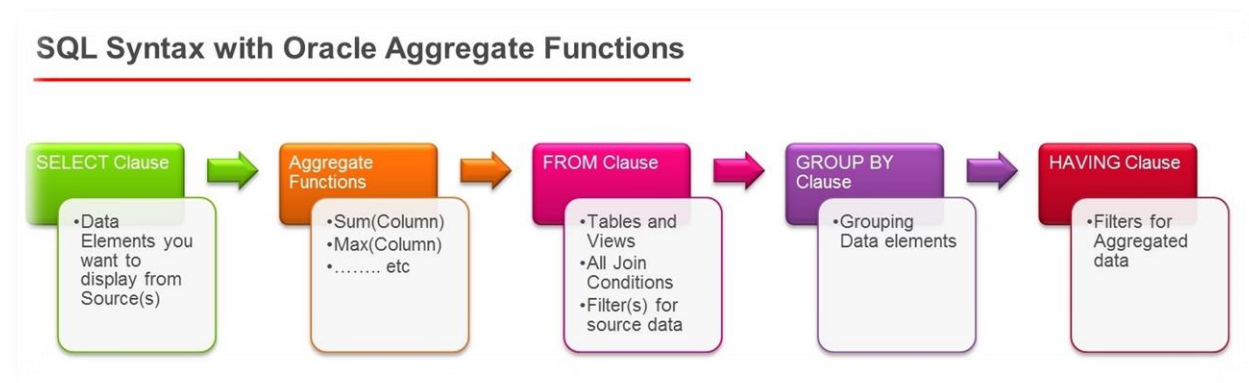


Figure 4.6: Working of Aggregate functions

Grouping

The SQL GROUP BY clause can be used in a SELECT statement to collect data across multiple records and group the results by one or more columns.

Ordering

The ORDER BY clause defines the ordering of rows based on columns from the SELECT clause. The ORDER BY clause is optional. If used, it must be the last clause in

the SELECT statement. ORDER BY sorts rows using the ordering columns in left-to-right, major-to-minor order.

ORDER BY column-1 [ASC|DESC] [column-2 [ASC|DESC]] ...

In Short,

1. FROM generates the data set
2. WHERE filters the generated data set
3. GROUP BY aggregates the filtered data set
4. HAVING filters the aggregated data set
5. SELECT transforms the filters aggregated data set
6. ORDER BY sorts the transformed data set
7. LIMIT .. OFFSET frames the sorted data set

Exercises

1. Choose all evil-masters with nationality as USA.
2. Identify the fictional master(s) who belong to England.
3. What is the highest level of evilness? Provide the description also.
4. Display all male minions with at least level-2 evilness.
5. Select all minions with level-3 evilness and hiring charge greater than 95.
6. Which minion is the oldest among all the other minions?
7. Which minion is the youngest among all the other minions?
8. What is the average hiring charge of the minions?
9. Which minion acts as a trainer for level-1 evilness?
10. Where mission 501 is going to happen? What is the estimate?
11. How many missions are in progress?
12. In which country the mission 504 happened?
13. Who is elected as the master for mission 502?
14. How many minions were involved in mission 502? Identify their names.
15. Identify the trainer and the student of training 403.
16. Display all missions that will happen between 10-JAN-2016 and 31-JAN-2016.
17. Identify all masters involved in any mission.
18. Display the currency of England.
19. Which currency have the highest conversion rate in terms of b-currency
20. Select a suitable minion to complete the mission 501

SQL JOIN CHEAT SHEET

QUERYING DATA FROM A TABLE

SELECT c1, c2 FROM t;

Query data in columns c1, c2 from a table

SELECT * FROM t;

Query all rows and columns from a table

SELECT c1, c2 FROM t

WHERE condition;

Query data and filter rows with a condition

SELECT DISTINCT c1 FROM t

WHERE condition;

Query distinct rows from a table

SELECT c1, c2 FROM t

ORDER BY c1 ASC [DESC];

Sort the result set in ascending or descending order

SELECT c1, c2 FROM t

ORDER BY c1

LIMIT n OFFSET offset;

Skip *offset* of rows and return the next *n* rows

SELECT c1, aggregate(c2)

FROM t

GROUP BY c1;

Group rows using an aggregate function

SELECT c1, aggregate(c2)

FROM t

GROUP BY c1

HAVING condition;

Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

SELECT c1, c2

FROM t1

INNER JOIN t2 ON condition;

Inner join t1 and t2

SELECT c1, c2

FROM t1

LEFT JOIN t2 ON condition;

Left join t1 and t2

SELECT c1, c2

FROM t1

RIGHT JOIN t2 ON condition;

Right join t1 and t2

SELECT c1, c2

FROM t1

FULL OUTER JOIN t2 ON condition;

Perform full outer join

SELECT c1, c2

FROM t1

CROSS JOIN t2;

Produce a Cartesian product of rows in tables

SELECT c1, c2

FROM t1, t2;

Another way to perform cross join

SELECT c1, c2

FROM t1 A

INNER JOIN t2 B ON condition;

Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

SELECT c1, c2 FROM t1

UNION [ALL]

SELECT c1, c2 FROM t2;

Combine rows from two queries

SELECT c1, c2 FROM t1

INTERSECT

SELECT c1, c2 FROM t2;

Return the intersection of two queries

SELECT c1, c2 FROM t1

MINUS

SELECT c1, c2 FROM t2;

Subtract a result set from another result set

SELECT c1, c2 FROM t1

WHERE c1 [NOT] LIKE pattern;

Query rows using pattern matching %, _

SELECT c1, c2 FROM t

WHERE c1 [NOT] IN value_list;

Query rows in a list

SELECT c1, c2 FROM t

WHERE c1 BETWEEN low AND high;

Query rows between two values

SELECT c1, c2 FROM t

WHERE c1 IS [NOT] NULL;

Check if values in a table is NULL or not

CODE SHEET

Select all tuples from a table

```
select * from minion;
```

Project some attributes from a table

```
select name, age from minion;
```

Project based on a condition

```
select name from minion where nationality = 'India';
```

select tuples based on a condition

```
select * from minion where hiring charge = 100;
```

select based on multiple conditions

```
select * from minion where gender = 'M' and age >6;
```

Use of aggregate function on an attribute

```
select max(age) from minion;
```

Use of aggregate function on a group

```
select evilness, max(age) from minion group by evilness;
```

Equijoin of two tables

```
select trainer from training t, takes_training tt  
where t.training_id = tt.training_id  
and t.level = 2;
```

Checking for NULL

```
select * from mission where from_date is null
```

```
select * from mission where from_date is not null
```

Order by

```
select name from minion order by age
```

Week 05

SQL Functions



Numeric Functions

Name	Description
ABS()*	Returns the absolute value of numeric expression
ACOS()*	Returns the arc-cosine of numeric expression. Returns NULL if the value is not in the range -1 to 1.
ASIN()*	Returns the arc-sine of numeric expression. Returns NULL if the value is not in the range -1 to 1.
ATAN()*	Returns the arc-tangent of numeric expression
ATAN2()*	Returns the arc-tangent of the two variables passed to it.
BIT_AND()*	Returns the bitwise AND all the bits in the expression.
BIT_COUNT()	Returns the string representation of the binary value passed to it.
BIT_OR()	Returns the bitwise OR of all the bits in the passed expression.
CEIL()*	Returns the smallest integer value that is not less than passed numeric expression.
CEILING()	Returns the smallest integer value that is not less than passed numeric expression.
CONV()	Convert numeric expression from one base to another.
COS()*	Returns the cosine of passed numeric expression. The numeric expression should be expressed in radians.
COT()	Returns the cotangent of passed numeric expression.
DEGREES()	Returns numeric expression converted from radians to degrees.
EXP()*	Returns the base of the natural logarithm(e) raised to the power of passed numeric expression.
FLOOR()*	Returns the largest integer value that is not greater than passed numeric expression.
FORMAT()	Returns a numeric expression rounded to a number of decimal places.
GREATEST()	Returns the largest value of the input expressions.

Name	Description
INTERVAL()	Takes multiple expressions exp1, exp2 and exp3 so on.. and returns 0 if exp1 is less than exp2, returns 1 if exp1 is less than exp3 and so on.
LEAST()	Returns the minimum-valued input when given two or more.
LOG()*	Returns the natural logarithm of the passed numeric expression.
LOG10()	Returns the base-10 logarithm of the passed numeric expression.
MOD()*	Returns the remainder of one expression by diving by another expression.
OCT()	Returns the string representation of the octal value of the passed numeric expression. Returns NULL if passed value is NULL.
PI()	Returns the value of pi.
POW()	Returns the value of one expression raised to the power of another expression.
POWER()*	Returns the value of one expression raised to the power of another expression.
RADIANS()	Returns the value of passed expression converted from degrees to radians.
ROUND()*	Returns numeric expression rounded to an integer. Can be used to round an expression to a number of decimal points.
SIN()*	Returns the sine of numeric expression given in radians.
SQRT()*	Returns the non-negative square root of numeric expression.
STD(),STDDEV()	Returns the standard deviation of the numeric expression.
TAN()*	Returns the tangent of numeric expression expressed in radians.
TRUNCATE()	Returns the tangent of numeric expression expressed in radians.

* – Supported in Oracle

String Functions

Name	Description
ASCII()*	Returns numeric value of leftmost character.
BIN()	Returns a string representation of the argument.
BIT_LENGTH()	Returns length of argument in bits.
CHAR_LENGTH()	Returns length of characters in argument.
CHAR()	Returns the character for each integer passed.
CONCAT_WS()	Returns concatenate with separator.
CONCAT()*	Returns concatenated string.
CONV()	Converts numbers between different bases.
ELT()	Returns the string at index number.
EXPORT_SET()	Returns a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string.
FIELD()	Returns the index (position) of the first argument in the subsequent arguments.
FIND_IN_SET()	Returns the index position of the first argument within the second argument.
FORMAT()	Returns a number formatted to specified number of decimal places.
HEX()	Returns a string representation of a hex value.
INSERT()	Inserts a substring at the specified position up to the specified number of characters.
INSTR()*	Returns the index of the first occurrence of substring.
LEFT()	Returns the leftmost number of characters as specified.
LENGTH()	Returns the length of a string in bytes.
LOAD_FILE()	Loads the named file.
LOCATE()	Returns the position of the first occurrence of substring.
LOWER()*	Returns the argument in lowercase.
LPAD()*	Returns the string argument, left-padded with the specified string.
LTRIM()*	Removes leading spaces.
MAKE_SET()	Returns a set of comma-separated strings that have the corresponding bit in bits set.
MID()	Returns a substring starting from the specified position.
OCT()	Returns a string representation of the octal argument.

Name	Description
ORD()	If the leftmost character of the argument is a multi-byte character, returns the code for that character.
QUOTE()	Escapes the argument for use in an SQL statement.
REGEXP()	Pattern matching using regular expressions.
REPEAT()	Repeats a string the specified number of times.
REPLACE()*	Replaces occurrences of a specified string.
REVERSE()*	Reverses the characters in a string.
RIGHT()	Returns the specified rightmost number of characters.
RPAD()*	Appends string the specified number of times.
RTRIM()*	Removes trailing spaces.
SOUNDEX()*	Returns a soundex string.
SOUNDS LIKE()	Compares sounds.
SPACE()	Returns a string of the specified number of spaces.
STRCMP()	Compares two strings.
SUBSTRING_INDEX()	Returns a substring from a string before the specified number of occurrences of the delimiter.
SUBSTRING(), SUBSTR()*	Returns the substring as specified.
TRIM()*	Removes leading and trailing spaces.
UNHEX()	Converts each pair of hexadecimal digits to a character.
UPPER()*	Converts to uppercase.

* – Supported in Oracle

Date/Time Functions

Name	Description
ADDDATE()	Adds dates
ADDTIME()	Adds time
CONVERT_TZ()	Converts from one time zone to another
CURDATE()	Returns the current date
CURTIME()	Returns the current time
DATE_ADD()	Adds two dates
DATE_FORMAT()	Formats date as specified
DATE_SUB()	Subtracts two dates
DATE()	Extracts the date part of a date or datetime expression
DATEDIFF()	Subtracts two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Returns the name of the weekday
DAYOFMONTH()	Returns the day of the month (1-31)
DAYOFWEEK()	Returns the weekday index of the argument
DAYOFYEAR()	Returns the day of the year (1-366)
EXTRACT()	Extracts part of a date
FROM_DAYS()	Converts a day number to a date
FROM_UNIXTIME()	Formats date as a UNIX timestamp
HOUR()	Extracts the hour
LAST_DAY()	Returns the last day of the month for the argument
LOCALTIME(), LOCALTIME	Synonym for NOW()
MAKEDATE()	Creates a date from the year and day of year
MICROSECOND()	Returns the microseconds from argument
MINUTE()	Returns the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Returns the name of the month
NOW()	Returns the current date and time
PERIOD_ADD()	Adds a period to a year-month
PERIOD_DIFF()	Returns the number of months between periods

Name	Description
QUARTER()	Returns the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Returns the second (0-59)
STR_TO_DATE()	Converts a string to a date
SUBDATE()	When invoked with three arguments a synonym for DATE_SUB()
SUBTIME()	Subtracts times
SYSDATE()	Returns the time at which the function executes
TIME_FORMAT()	Formats as time
TIME_TO_SEC()	Returns the argument converted to seconds
TIME()	Extracts the time portion of the expression passed
TIMEDIFF()	Subtracts time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression. With two arguments, the sum of the arguments
TIMESTAMPADD()	Adds an interval to a datetime expression
TIMESTAMPDIFF()	Subtracts an interval from a datetime expression
TO_DAYS()	Returns the date argument converted to days
UNIX_TIMESTAMP()	Returns a UNIX timestamp
UTC_DATE()	Returns the current UTC date
UTC_TIME()	Returns the current UTC time
UTC_TIMESTAMP()	Returns the current UTC date and time
WEEK()	Returns the week number
WEEKDAY()	Returns the weekday index
WEEKOFYEAR()	Returns the calendar week of the date (1-53)
YEAR()	Returns the year
YEARWEEK()	Returns the year and week

EXERCISE

1. Find the ASCII values of the masters' names.
2. Append the country names in front of the minion names.
3. Ceil the currency values which start with a consonant and floor the others.

4. Repeat the string 'ba' a dozen number of times.
5. Reverse the names of all the minions.
6. Convert all the country names to uppercase.
7. Display the current time.
8. Find the minion with least salary
9. How long (in weeks) was the mission 504.
10. Display the duration of all training offered.

Week 06

Set Operators

The results of two or more SELECT statements can be combined together using set operators under two conditions (called as Union-Compatible).

- (i) The arity of the relation R and S to be same.
- (ii) The domain of each attribute in the column order to be same in both R and S

OPERATOR	DESCRIPTION	DUPLICATES
UNION	Returns the combined results of SELECT statements.	NO
UNION ALL	Returns the combined results of SELECT statements along with duplicates.	YES
INTERSECT	Returns only the results common to the SELECT statements	NO
MINUS	Removes the second query's results from the first query's results.	NO

Table 6.1: Set Operators

Degree of precedence : Same for all operators

Evaluation : Left to Right/Top to Bottom

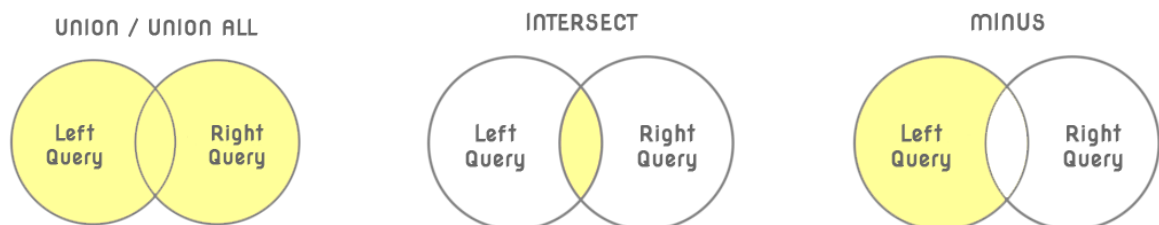


Figure 6.1: Set Operators

Union

UNION is used to combine the results of two or more Select statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables.

```
SELECT column_name(s)
FROM table1
[WHERE conditions][order by col asc|desc]
UNION
SELECT column_name(s)
FROM table2
[WHERE conditions][order by col asc|desc];
```

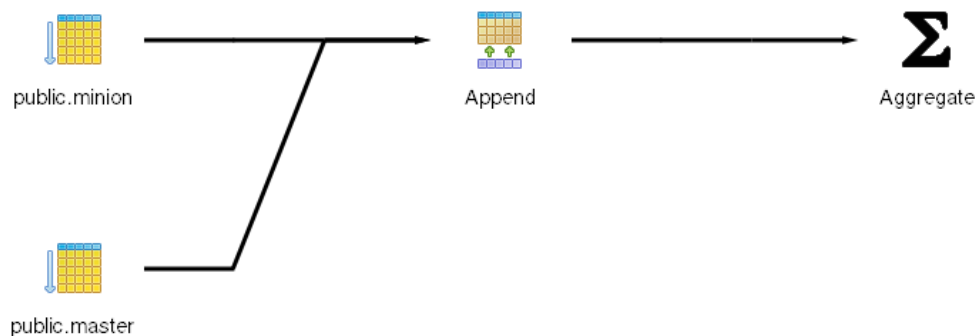


Figure 6.2: Using Union to find all minions and masters

The figure 6.2 depicts the union operation on two relations minion and master. As the attributes are not similar across the two tables, to satisfy the Union-compatible property, projection of attributes is done. In minion relation, minion_id and name is projected; whereas in master relation, master_id and name is projected. Union-completeness property is satisfied as,

- (1) Both projected relations have the arity 2
- (2) The domain of the attribute in column order in both relation is number followed by varchar.

Hence, on applying the union set operator, 22 rows (which is an aggregate of 12 minions and 10 masters) will be obtained.

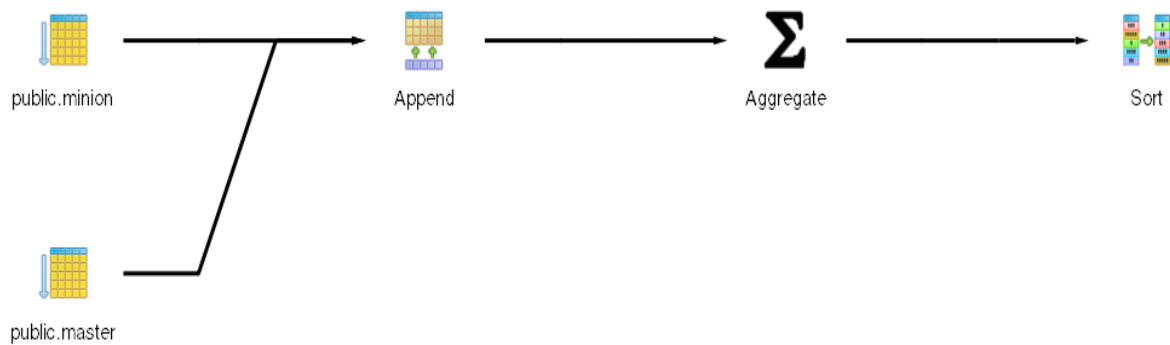


Figure 6.3: Sort followed by Union operation

It could be noted during execution that, the values of the attributes of the first select statement will be appended by the tuples (from the subsequent select statements) that satisfies the conditions. Hence, order by clause would work on the attributes of the first select statement as shown in Figure 6.3.

Union All

This operation is similar to Union. But it also shows the duplicate rows.

```

SELECT column_name(s) FROM table1 [WHERE conditions]
UNION ALL
SELECT column_name(s) FROM table2 [WHERE conditions];

```

Intersect

INTERSECT operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of Intersect the number of columns and datatype must be same. MySQL does not support INTERSECT operator.

```

SELECT column_name(s) FROM table1 [WHERE conditions]
INTERSECT
SELECT column_name(s) FROM table2 [WHERE conditions];

```

Figure 6.4 depicts the intersection operation between two relations minion and training (w.r.t the attribute minion_id). The projection of tuples in both relation is

done in Subquery scan. All the tuples (of both relations) will be appended together and then will result is obtained after the Hash Intersection operation.

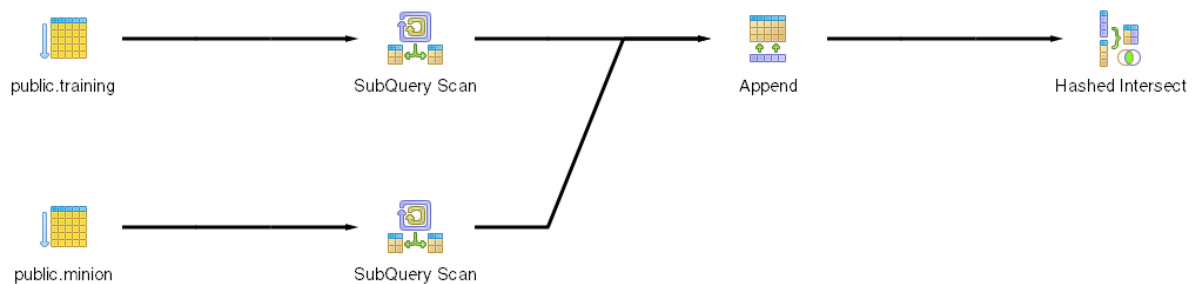


Figure 6.4: Intersect Operation

Minus

MINUS operation combines result of two Select statements and return only those results which belong to first set of result. MySQL does not support INTERSECT operator.

```

SELECT column_name(s) FROM table1 [WHERE conditions]
MINUS
SELECT column_name(s) FROM table2 [WHERE conditions];
  
```

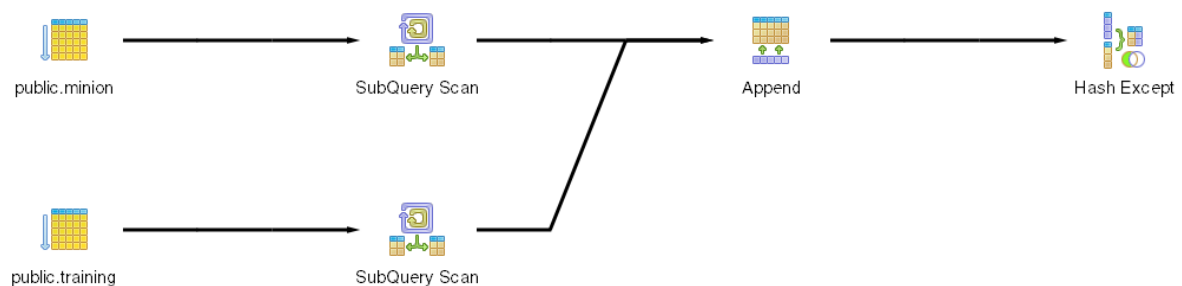


Figure 6.5: Minus Operation

Similar to Intersect operation in Minus, after the tuples are appended (satisfying the Union-compatibility), Hash Except operation is done to produce the intended result.

Exercises:

Note: Use Set Operations only to solve the questions given below.

1. Find all minions who did not undergo training currently.
2. Select all masters from Japan who are involved in any mission.
3. Identify the list of minions and masters who share the same nationality.
4. How many minions do not train other minions?
5. Identify the trainers who are also involved in any mission.
6. Find the average hiring charge of the successfully trained minions.
7. Which evilness does the 8 year old minion(s) of level 3 or 4 possess?
8. What kind of evilness a minion will possess if it's level is 4 or it's age is greater than 7.
9. Identify the mission in which minions are yet to be involved.
10. Select all missions to which minions are assigned to.
11. Which mission has made payment to the minions upon completion of the mission?
12. Which female minion was involved in a mission and received payment.
13. Which female minion was involved in a mission and yet to receive payment?
14. Isolate the count of male and female minions, who have participated in the mission and not received payment
15. Find all minions involved in mission and does not undergo training currently; but they teach any course.

CODE SHEET

#UNION OF TWO RELATION

```
select minion_id,name
from minion
union
select master_id,name
from master
```

#USE OF WHERE CLAUSE

```
select minion_id,name
from minion
where nationality = 'usa'
union
select master_id,name
from master
where nationality = 'japan'
```

#USE OF ORDER BY

```
select minion_id,name
from minion
where nationality = 'usa'
union
select master_id,name
from master
where nationality = 'japan'
order by name
```

#NESTED QUERIES

```
(select minion_id from minion
union
select master_id from master)
except
(select minion_id from minion_involved_in_mission
union
select master_id from minion_involved_in_mission)
```

#USE OF ANY/SOME IN WHERE CONDITION

```
select name, nationality
from minion
where nationality = any (
    select nationality from minion
);
```

#USE OF ALL IN WHERE CONDITION

```
select mission_id
from minion_involved_in_mission
```

```
where minion_id > all(301,306);
```

note: syntax for postgres

```
select mission_id  
from minion_involved_in_mission  
where minion_id > all('{301,306}'::int[])
```

#USE OF SOME IN WHERE CONDITION

```
select name  
from minion  
where age < some(9,5);
```

```
select name  
from minion  
where age < some(9,5)  
and name like '_i%'
```

```
select name  
from minion  
where nationality < some('usa','japan')  
and gender like 'm%'
```

note: syntax for postgres

```
select name  
from minion  
where nationality < some('{usa,japan}'::text[])  
and gender like 'm%'
```

Week 07

Revisiting date time Functions

This week we deal with storing and retrieving dates, converting to other date formats, set a default date format and perform functions with date. As we know the default date format is DD-MON-YYYY, where

DD is a two-digit day such as 26

MON is the first three letters of the month such as APR

YYYY is a four-digit year such as 1984

Oracle has functions that enable you to convert a value in one data type to another.

Function	Description
TO_CHAR(x [, format])	Converts, the number or date-time x to a string. You can also supply an optional format for x.
TO_DATE(x [, format])	Converts the string x to a DATE.
ADD_MONTHS(x, y)	Returns the result of adding y months to x. If y is negative, y months are subtracted from x
LAST_DAY(x)	Returns the last day of the month that contains x
MONTHS_BETWEEN(x, y)	Returns the number of months between x and y. If x appears before y on the calendar, the number returned is positive, otherwise the number is negative.
NEXT_DAY(x, day)	Returns the datetime of the next day following x; day is specified as a literal string-THURSDAY, for example.
ROUND(x [, unit])	Rounds x. By default, x is rounded to the beginning of the nearest day.

Function	Description
	<p>You may supply an optional unit string to indicate the rounding unit.</p> <p>For example, <code>YYYY</code> rounds <code>x</code> to the first day of the nearest year.</p>
<code>SYSDATE()</code>	<code>SELECT SYSDATE FROM dual;</code>
<code>TRUNC(x [, unit])</code>	<p>Truncates <code>x</code>. By default, <code>x</code> is truncated to the beginning of the day.</p> <p>You may supply an optional unit string that indicates the truncating unit. For example, <code>MM</code> truncates <code>x</code> to the first day of the month.</p>

Exercises

1. Display the starting date of mission in 'MON DD, YYYY' format.
 2. In 'DD-MM-YY' format, display the completion date of training trained by Bob.
 3. In which month Phil will complete his training.
 4. After each training period, the trainers take vacation for a week.
 - a. So, when would the next batch start for level 1?
 - b. When did the previous batch for level 2 would have commenced?
 - c. To which upcoming batch of level 3 can minion 306 enroll?
- (Hint: In ADD_MONTHS, If y is negative, y months are subtracted from x.)
5. Find the last day of the month on which mission 502 have completed.
 6. Update the starting date of mission 502 as 16-May-2017. Find the months between the minion 301 (Stuart) have completed its training (id 401) and involved in a mission.
 7. Round the date of payment w.r.t month for the mission 504.
 8. Truncate the date of payment w.r.t month for the mission 504.
 9. Round off the completion date of the all missions involving Stuart or Lance.
 10. Find the master who spent the most through missions.

CODE SHEET

To display the date in other formats

```
SELECT customer_id, TO_CHAR(dob, 'MONTH DD,YYYY') FROM customers;
```

To display the Timestamp in other formats

```
SELECT TO_CHAR(SYSDATE, 'MONTH DD, YYYY,HH24:MI:SS') FROM dual;
```

To add a specific count of days to a date

```
SELECT ADD_MONTHS('01-JAN-2005', 13) FROM dual;
```

To identify the last day

```
SELECT LAST_DAY('01-JAN-2005') FROM dual;
```

To calculate the months between two dates

```
SELECT MONTHS_BETWEEN('25-MAY-2005', '15-JAN-2005') FROM dual;
```

To identify the next Saturday from the given date

```
SELECT NEXT_DAY('01-JAN-2005', 'SATURDAY') FROM dual;
```

To round off to the nearest date

```
SELECT ROUND(TO_DATE('25-OCT-2005'), 'YYYY')FROM dual;
```

To truncate the given date to the beginning of the day w.r.t the unit specified

```
SELECT TRUNC(TO_DATE('25-MAY-2005'), 'YYYY')FROM dual;
```

To display the current system time

```
SELECT SYSDATE FROM dual;
```


Sub-queries

Subqueries are queries that run inside other queries or statements, like data manipulation statements – INSERT, UPDATE, and DELETE. The query or statement that contains a subquery is also known as an outer query, and subqueries are inner queries.

There are two basic types of subqueries:

Single row subqueries Return zero or one row to the outer SQL statement.

Multiple row subqueries Return one or more rows to the outer SQL statement.

In addition, there are three subtypes of subqueries that may return single or multiple rows:

Multiple column subqueries Return more than one column to the outer SQL statement.

To handle a subquery that returns multiple rows, your outer query may use the IN, ANY, or ALL operator. As you saw before, you can use these operators to check and compare values supplied in a list of literal values.

Correlated subqueries Reference one or more columns in the outer SQL statement. The subquery is known as a correlated subquery because the subquery is related to the outer SQL statement.

Nested subqueries Are placed within another subquery. You can nest subqueries to a depth of 255.

Exercises:

1. [Subquery in Where clause] Identify all minions along with their hiring_charges involved in a mission with estimate greater than 2000.
2. [Subquery in Where clause] Find all minions paid in Indian currency (INR).
3. [Subquery in Having clause] Find the name and the hiring_charge of the minions with charges more than the average hiring_charge of the minions.
4. [Subquery in Having clause] Identify the trainers for mission 503.
Hint: level n trainer can teach level n-1. Vice versa is not true
5. [Subquery in From clause] Find the Indian minions trained by Steve or Mike.
6. [Subquery in From clause] Which master from the cartoon genre have completed mission outside India.
7. [Correlated Subquery] Find the missions with estimate higher than the average estimate of all missions.
8. [Correlated Subquery] Compare the duration of all training and find all occurrences except the training with least duration.

CODE SHEET

Subqueries in 'Where' clause

```
select minion_id, name
from minion
where minion_id in (select trainer from training t)
```

Subqueries in Having clause

```
select    hiring_charge, count(*)
from      minion
group by  hiring_charge
having    hiring_charge < (select avg(hiring_charge) from minion);
```

Subqueries in From clause

```
select distinct name
from master, (select master_id, mission_id from
minion_involved_in_mission) as selected_involved
where selected_involved.master_id = master.master_id
```

```
select name, nationality, pre_req, level, hiring_charge from
(select pre_req, country
from mission where mission_id in(502,504)) as selected_mission,
training, minion
where selected_mission.pre_req = training.training_id
and selected_mission.country = minion.nationality
```

Subqueries resulting more than one column

```
select * from minion where (minion_id, nationality) in
(select mim.minion_id, ma.country
from mission ma, minion_involved_in_mission mim
where ma.mission_id = mim.mission_id
and mim.mission_id = 502);
```

Correlated Subqueries

```
SELECT mo.minion_id, name, nationality, hiring_charge
FROM minion mo
WHERE hiring_charge > (SELECT AVG(hiring_charge) FROM minion mi
                       WHERE mo.minion_id = mi.minion_id);
```

Week 08

A view is a virtual table based on the result-set of an SQL statement. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, allow users to do the following:

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views:

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables, or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows:

Creating a View

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL SELECT query.

WITH CHECK OPTION:

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

Creating a View with CHECK option :

```
CREATE VIEW MINION_VIEW AS
SELECT name, age
FROM MINION
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View:

A view can be updated under certain conditions:

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So if a view satisfies all the above-mentioned rules then you can update a view.

Creating a View with CHECK option :

```
UPDATE MINION_VIEW
SET AGE = 9
WHERE name='Dave';
```

This would ultimately update the base table MINION and same would reflect in the view itself. Now, try to query base table and analyse.

Inserting Rows into a View:

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here we can not insert rows in MINION_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in similar way as you insert them in a table.

Deleting Rows into a View:

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE= 22.

Deleting from a View:

```
DELETE FROM MINION_VIEW  
WHERE AGE = 5;
```

This would ultimately delete a row from the base table MINION and same would reflect in the view itself. Now, try to query base table, and analyse.

Dropping Views:

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple as given below:

Subqueries in 'Where' clause

```
DROP VIEW view_name;
```

CODE SHEET

Create a View

```
create view view_name  
as  
<any select command>
```

Replace a View

```
create or replace view view_name  
as  
<any select command>
```

Rename a view

```
rename <old_view> to <new_view>
```

Drop a view

```
drop view view_name
```

Insert into View

```
Insert into view_name values (...)
```

```
Insert into view_name (col names...) values (...) [where condition]
```

Update a view

```
update view_name set col = value where condition
```

Delete from a view

```
delete from view_name where conditions
```

Exercises

1. Find all minions who have trained more than one minion.
2. Identify and list the first three currencies that have the highest currency conversion rate.
3. To extract the male minions of Japan or England who are committed in any mission.
4. Show the evilness description of all minions along with their hiring charge.
5. Which minion(s) are associated with any mission currently? Who are their masters?
6. Find all masters whose mission(s) were a success.
7. Create a savepoint.
8. Add a trainer to the relation and check whether the view (created in Question 1) gets updated.
9. Add a male minion of Japan and check whether the view (created in Question 3) gets updated.
10. Create a updatable view for payment history.

Week 09

PL/SQL

PL/SQL stands for Procedural Language extensions to the Structured Query Language (SQL). SQL is a powerful language for both querying and updating data in relational databases. Oracle created PL/SQL that extends some limitations of SQL to provide a more comprehensive solution for building mission-critical applications running on Oracle database.

PL/SQL BLOCK

PL/SQL program units organize the code into blocks. A block without a name is known as an anonymous block. The anonymous block is the simplest unit in PL/SQL. It is called anonymous block because it is not saved in the Oracle database. An example of an anonymous block is shown below.

```
[DECLARE]
    Declaration statements;
BEGIN
    Execution statements;
[EXCEPTION]
    Exception handling statements;
END;
/
```

Variable Declaration

Following are the naming conventions

- The variable name must be less than 31 characters.
- The variable name must begin with an ASCII letter.
- Any count of '_' and '\$' could be used.
- Meaningful name for variables. Use prefixes for variables. [optional]

○ v_ for varchar datatype	○ t_ for table
○ n_ for number	○ r_ for row
○ d_ for date and so on.	○ b_ for boolean

Variable Anchors

In PL/SQL program, one of the most common tasks is to select values from columns in a table into a set of variables. In case the data types of columns of the table changes, you have to change the PL/SQL program to make the types of the variables compatible with the new changes.

PL/SQL provides you with a very useful feature called variable anchors. It refers to the use of the **%TYPE keyword** to declare a variable with the data type is associated with a column's data type of a particular column in a table. Example shown below.

```
DECLARE
    v_name <table_name>.<column_name>%TYPE;
BEGIN
END;
/
```

Variable Assignment

Assignment operator in PL/SQL is :=

The example below shows Zugi is assigned to v_name.

```
DECLARE
    v_name <table_name>.<column_name>%TYPE;
BEGIN
    v_name := 'Zugi';
END;
/
```

PL/SQL RECORD

A PL/SQL record is a composite data structure that is a group of related data stored in fields. Each field in the PL/SQL record has its own name and data type.

Declaring Table-based Record

To declare a table-based record you use a table name with `%ROWTYPE` attribute. The fields of the PL/SQL record has the same name and data type as the column of the table.

```
DECLARE

    table_based_record table_name%ROWTYPE;
```

User Defined Record

A record could be defined explicitly. A record may have many fields.

```
TYPE type_name IS RECORD

    (field1 data_type1 [NOT NULL] := [DEFAULT VALUE],

    field2 data_type2 [NOT NULL] := [DEFAULT VALUE],

    ...

    fieldn data_type3 [NOT NULL] := [DEFAULT VALUE]

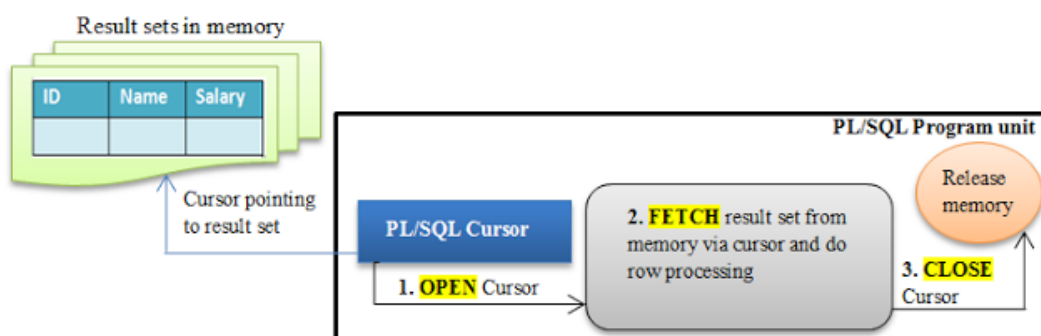
    );
```

Introduction to PL/SQL Cursor

When you work with Oracle database, you work with a complete set of rows returned from an SQL SELECT statement. However the application in some cases cannot work effectively with the entire result set, therefore, the database server needs to provide a mechanism for the application to work with one row or a subset of the result set at a time. As the result, Oracle created PL/SQL cursor to provide these extensions.

A PL/SQL cursor is a pointer that points to the result set of an SQL query against database tables.

WORKING OF PL/SQL CURSOR



Declaring PL/SQL Cursor

To use PL/SQL cursor, first you must declare it in the declaration section of PL/SQL block or in a package as follows:

```
CURSOR cursor_name [ ( [ parameter_1 [, parameter_2 ...] ) ]
[ RETURN return_specification ]
IS sql_select_statements
[FOR UPDATE [OF [column_list]]];
```

Opening a PL/SQL Cursor

After declaring a cursor, you can open it by using the following syntax:

```
OPEN cursor_name [ ( argument_1 [, argument_2 ...] ) ];
```

You have to specify the cursor's name `cursor_name` after the keyword `OPEN`. If the cursor was defined with a parameter list, you need to pass corresponding arguments to the cursor.

When you `OPEN` the cursor, PL/SQL executes the SQL `SELECT` statement and identifies the active result set. Notice that the `OPEN` action does not actually retrieve records from the database. It happens in the `FETCH` step. If the cursor was declared with the `FOR UPDATE` clause, PL/SQL locks all the records in the result set.

Fetching Records from PL/SQL Cursor

Once the cursor is open, you can fetch data from the cursor into a record that has the same structure as the cursor. Instead of fetching data into a record, you can also fetch data from the cursor to a list of variables.

The fetch action retrieves data and fills the record or the variable list. You can manipulate this data in memory. You can fetch the data until there is no record found in active result set.

The syntax of `FETCH` is as follows:

```
FETCH cursor_name INTO record or variables
```

You can test the cursor's attribute `%FOUND` or `%NOTFOUND` to check if the fetch against the cursor is succeeded.

We can use PL/SQL `LOOP` statement together with the `FETCH` to loop through all records in active result set (refer code sheet).

Closing PL/SQL Cursor

You should always close the cursor when it is no longer used. Otherwise, you will have a memory leak in your program, which is not expected.

```
CLOSE cursor_name;
```

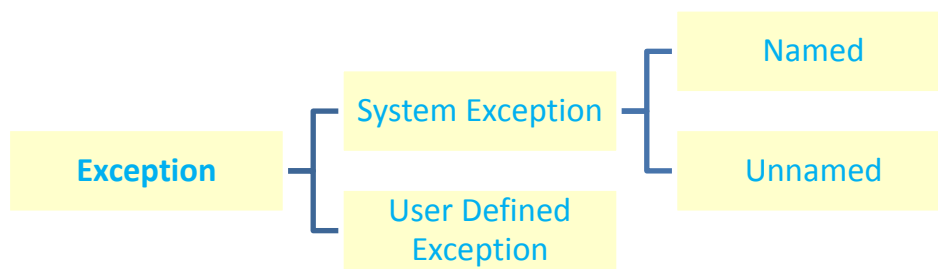
PL/SQL Cursor Attributes

These are the main attributes of a PL/SQL cursor and their descriptions.

ATTRIBUTE	DESCRIPTION
cursor_name%FOUND	returns TRUE if record was fetched successfully by cursor cursor_name
cursor_name%NOTFOUND	returns TRUE if record was not fetched successfully by cursor cursor_name
cursor_name%ROWCOUNT	returns the number of records fetched from the cursor cursor_name at the time we test %ROWCOUNT attribute
cursor_name%ISOPEN	returns TRUE if the cursor cursor_name is open

Introduction to PL/SQL Exceptions

In PL/SQL, any kind of errors is treated as exceptions. An exception is defined as a special condition that change the program execution flow. There are two types of exceptions.



Named System Exceptions are exceptions that have been given names by PL/SQL. They are named in the STANDARD package in PL/SQL and do not need to be defined by the programmer.

Oracle Exception Name	Oracle Error	Explanation
DUP_VAL_ON_INDEX	ORA-00001	You tried to execute an INSERT or UPDATE statement that has created a duplicate value in a field restricted by a unique index.
TIMEOUT_ON_RESOURCE	ORA-00051	You were waiting for a resource and you timed out.
TRANSACTION_BACKED_OUT	ORA-00061	The remote portion of a transaction has rolled back.
INVALID_CURSOR	ORA-01001	You tried to reference a cursor that does not yet exist. This may have happened because you've executed a FETCH cursor or CLOSE cursor before OPENing the cursor.

Oracle Exception Name	Oracle Error	Explanation
NOT_LOGGED_ON	ORA-01012	You tried to execute a call to Oracle before logging in.
LOGIN_DENIED	ORA-01017	You tried to log into Oracle with an invalid username/password combination.
NO_DATA_FOUND	ORA-01403	You tried one of the following: <ol style="list-style-type: none"> 1. You executed a SELECT INTO statement and no rows were returned. 2. You referenced an uninitialized row in a table. 3. You read past the end of file with the UTL_FILE package.
TOO_MANY_ROWS	ORA-01422	You tried to execute a SELECT INTO statement and more than one row was returned.
ZERO_DIVIDE	ORA-01476	You tried to divide a number by zero.
INVALID_NUMBER	ORA-01722	You tried to execute a SQL statement that tried to convert a string to a number, but it was unsuccessful.
STORAGE_ERROR	ORA-06500	You ran out of memory or memory was corrupted.
PROGRAM_ERROR	ORA-06501	This is a generic "Contact Oracle support" message because an internal problem was encountered.
VALUE_ERROR	ORA-06502	You tried to perform an operation and there was a error on a conversion, truncation, or invalid constraining of numeric or character data.
CURSOR_ALREADY_OPEN	ORA-06511	You tried to open a cursor that is already open.

Unnamed System Exceptions are exceptions, for which Oracle does not provide any name. These type of exception do not occur frequently. Unnamed exceptions are handled in two ways.

- Using WHEN OTHERS exception handler
- By associating a code and name and convert to a named exception.

Syntax

```
DECLARE
[declaration_section]
    exception_name EXCEPTION;
    PRAGMA
    EXCEPTION_INIT(err_name, err_code);

BEGIN
    executable_section
    RAISE exception_name;

EXCEPTION
    WHEN exception_name THEN
        [statements]

    WHEN OTHERS THEN
        [statements]

END [procedure_name];
```

User-Defined Exceptions

Apart from system exceptions, the constraints related to business logic are handled using user-defined exceptions.

CODE SHEET

Create a Block

```
SET SERVEROUTPUT ON SIZE 1000000
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello PL/SQL');  --to display
END;
/
```

Using %TYPE

```
DECLARE
    v_name_of_minion minion.name%TYPE;
    n_age_of_minion minion.age%TYPE;
BEGIN
END;
/
```

Initializing Variables

```
DECLARE
    v_name_of_minion minion.name%TYPE;
    n_age_of_minion minion.age%TYPE;
BEGIN
    v_name_of_minion := 'Stuart';
    n_age_of_minion  := 6;
END;
/
```

Using %ROWTYPE

```
SET SERVEROUTPUT ON SIZE 1000000;
DECLARE
    r_mini minion%ROWTYPE;
    n_minion_id minion.minion_id%TYPE := 305;
BEGIN
    SELECT *
    INTO r_mini
    FROM minion
    WHERE minion_id = n_minion_id;
    DBMS_OUTPUT.PUT_LINE(r_mini.nationality);
END;
/
```

In the above example:

- First, we defined a record, based on minion table.
- Second, we used the SELECT statement to retrieve the information of the minion with id 305 and populate the data into the r_mini record.
- Third, we print out the nationality of the selected minion from the r_mini record.

Defining Custom ROWTYPE

-- This block is to find all underway mission happening in Japan.

```
SET SERVEROUTPUT ON SIZE 1000000;
DECLARE
    TYPE mission_completion IS RECORD(
        v_id mission.mission_id%TYPE,
        v_country mission.country_id%TYPE
    );
    r_mission_completion_in_japan mission_completion; -- name record
    v_status missin.status%TYPE := 'underway';

BEGIN
    SELECT mission_id, country
    INTO r_mission_completion_in_japan
    FROM mission
    WHERE status = v_status
    AND country = 'Japan';
    -- print out the employee's name
    DBMS_OUTPUT.PUT_LINE(r_mission_completion_in_japan.v_id || ', ' ||
r_mission_completion_in_japan.v_country);
END;
/
```

Cursor Definition

-- to find all trainers

```
CURSOR minion_trainer IS
SELECT name
FROM minion m, training t
WHERE m. minion_id = t.trainer;
```

We retrieved data from joining two tables, minion and training. Four tuples would be received (w.r.t trainers: 304, 312, 302 and 308). The four tuples are referred as result-set.

Opening a Cursor

```
OPEN minion_trainer
```

Fetching from Cursor

LOOP

```
FETCH minion_trainer INTO trainer; --trainer as like minion%ROWTYPE
```

```
EXIT WHEN minion_trainer%NOTFOUND;
```

```
DBMS_OUTPUT.PUT_LINE(trainer.minion_id|| ' - ' || trainer.name);
```

```
END LOOP;
```

Closing a Cursor

```
CLOSE minion_trainer
```

Exercises

1. Find the masters and minions of nationality England.
2. Identify the minion with lowest lowest hiring charge for evilness – Tickle.
3. Identify all mission that did not happen in Germany.
4. Find the duration of the training handled by Bob or Steve.
5. Calculate the amount in hand after the cancellation of mission 505. Assume each minion demands 10~~e~~ as mission cancellation charge.
6. Did mission 504 overrun the estimated amount?
7. Find the minion employed in
 - a. more than one mission
 - b. atleast one mission
 - c. not participated
8. Find the currency rate of INR with respect to all the other currencies.

Week 10

PL/SQL Loops and Controls

To iterate a block of statements loops are used. Three types of loops in PL/SQL are

- Basic Loop

```
[ label_name ] LOOP
    statement(s);
END LOOP [ label_name ];
```

- While Loop

```
[ label_name ] WHILE condition LOOP
    statement(s);
END LOOP [ label_name ];
```

- For Loop

```
[ label_name ] FOR current_value IN [ REVERSE ]
lower_value..upper_value LOOP
    statement(s);
END LOOP [ label_name ];
Note: proceed from upper_value to lower_value range.
```

The **PL/SQL IF statement** allows you to execute a sequence of statements conditionally. The IF statement evaluates a condition. The condition can be anything that evaluates to a logical value of true or false such as comparison expression or a combination of multiple comparison expressions.

```
IF condition1 THEN
    sequence_of_statements1
ELSIF condition2 THEN
    sequence_of_statements2
ELSE
    sequence_of_statements3
END IF;
```

The **PL/SQL CASE statement** allows you to execute a sequence of statements based on a selector. A selector can be anything such as variable, function, or expression that the CASE statement evaluates to a Boolean value.

PL/SQL selector based CASE statement

```
[<<label_name>>]
CASE [TRUE | selector]
    WHEN expression1 THEN
        sequence_of_statements1;
    WHEN expression2 THEN
        sequence_of_statements2;
    ...
    WHEN expressionN THEN
        sequence_of_statementsN;
    [ELSE sequence_of_statementsN+1;]
END CASE [label_name];
```

PL/SQL searched CASE statement

```
[<<label_name>>]
CASE
    WHEN search_condition_1 THEN sequence_of_statements_1;
    WHEN search_condition_2 THEN sequence_of_statements_2;
    ...
    WHEN search_condition_N THEN sequence_of_statements_N;
    [ELSE sequence_of_statements_N+1;]
END CASE [label_name];
```

CODE SHEET

Using loop to iterate

```
DECLARE
    no NUMBER := 5;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE ('Inside value:  no = ' || no);
        no := no -1;
        IF no = 0 THEN
            EXIT;
        END IF;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Outside loop end');
END;
/
```

Using while loop to iterate

```
DECLARE
    no NUMBER := 0;
BEGIN
    WHILE no < 10 LOOP
        no := no + 1;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Sum :' || no);
END;
/
```

Using for loop to iterate

```
BEGIN
    FOR no IN 1 .. 5 LOOP
        DBMS_OUTPUT.PUT_LINE('Iteration : ' || no);
    END LOOP;
END;
/
```


Using Case Statement : Example 1

```
DECLARE
  n_nationality minion.nationality%TYPE;
  n_minion_id minion.minion_id%TYPE := 304;
  v_description varchar(30);
BEGIN
  -- get commission percentage
  SELECT nationality
  INTO n_nationality
  FROM minion
  WHERE minion_id = n_minion_id;
  CASE n_nationality

    WHEN 'USA' THEN
      v_description:= ' belongs to United States';
    WHEN 'Germany' THEN
      v_description:= ' belongs to Germany';
    ELSE
      v_description:= ' does not belong to either US or Germany';
  END CASE;
  DBMS_OUTPUT.PUT_LINE('Minion ' || minion_id || v_description);
END;
```

/

Output:

Minion 304 does not belong to either US or Germany

Using Case Statement : Example 2

```
DECLARE
  n_hiring_charge minion.hiring_charge%TYPE;
  n_minion_id minion.minion_id%TYPE := 305;
BEGIN
  SELECT hiring_charge
  INTO n_hiring_charge
  FROM minion
  WHERE minion_id = n_minion_id;

  CASE
    WHEN n_hiring_charge < 100 THEN
      v_msg := 'Hiring charge is less than 100';
    WHEN n_hiring_charge >= 100 THEN
      v_msg := 'Hiring charge is greater than 100';
  END CASE;

  DBMS_OUTPUT.PUT_LINE(v_msg);
END;
```

Using System Defined Exception

```
DECLARE
    temp minion%ROWTYPE;
BEGIN
    SELECT * INTO temp FROM minion
        WHERE minion_id > 314;
EXCEPTION
    WHEN no_data_found THEN
        dbms_output.put_line("No data for the given input");
END;
/
```

Using User Defined Exception

```
DECLARE
    myex EXCEPTION;
    n NUMBER := &n;
BEGIN
    FOR i IN 1..n LOOP
        dbms_output.put_line(i);
        IF i=n THEN
            RAISE myex;
        END IF;
    END LOOP;
EXCEPTION
    WHEN myex THEN
        RAISE_APPLICATION_ERROR(-20015, 'loop finish');
END;
/
```

Exercises

1. Create a temporary table "OddEven" with three attributes - index, num, type. Datatype of the attributes are number, number and varchar respectively. Create a PL/SQL block that takes one number at a time (from 100 to 125), deduce whether it is odd or even number and update the same in the "type" attribute of the temporary table.

Index	Num	Type
1	100	Even
2	101	Odd
..
26	125	Odd

2. Create a PL/SQL block to display all the minions of age < 5.
3. Display the evilness (description) of each minion using case statement.
4. Display the master and the country of all missions that require pre_req greater than three and missions not undergoing in USA.
5. Find all the minions with level 2 evilness. Using control structures display whether they are undergoing training for level 3 or not.

PL/SQL Procedure

PL/SQL procedures create using CREATE PROCEDURE statement. The major difference between PL/SQL function or procedure, function return always value whereas procedure may or may not return value.

When you create a function or procedure, you have to define IN/OUT/INOUT parameters.

- **IN:** input parameter.
- **OUT:** output parameter.
- **IN OUT:** input and output parameter.

```
CREATE [OR REPLACE] PROCEDURE [SCHEMA..] procedure_name
    [ (parameter [,parameter]) ]
IS
    [declaration_section
        variable declarations;
        constant declarations;
    ]
BEGIN
    [executable_section
        PL/SQL execute/subprogram body
    ]
    [EXCEPTION]
        [exception_section
            PL/SQL Exception block
        ]
END [procedure_name];
/
```

```
DROP PROCEDURE procedure_name;
```

CODE SHEET

Creating a Procedure

```
CREATE or REPLACE PROCEDURE getMinionDetails(no in number,
t_minion out minion%rowtype)
IS
BEGIN
    SELECT * INTO t_minion FROM minion WHERE minion_id = no;
END;
/
```

Calling a Procedure

```
DECLARE
    temp_minion minion%rowtype;
    no number :=&no;
BEGIN
    getMinionDetails(no,temp);
    dbms_output.put_line('Details of Minion' || ' ' ||
        temp_minion.minion_no || ' ' ||
        temp_minion.name || ' ' ||
        temp_minion.nationality || ' ' ||
        temp_minion.gender);
);
END;
/
```

Drop a Procedure

```
DROP PROCEDURE getMinionDetails;
```

Exercises

1. Create a procedure to display all fictional masters assigned with missions.
2. Create a procedure to display all female minions with evilness \leq 3 and undergoes training.
3. Create a procedure that gets the mission_id and find the corresponding mission status and the estimate.
4. Create a procedure that takes trainer as the input and displays the trainees.
5. Write a procedure to display all missions that happen in January.

PL/SQL Functions

PL/SQL functions block create using CREATE FUNCTION statement. The major difference between PL/SQL function or procedure, function return always value whereas procedure may or may not return value.

When you create a function or procedure, you have to define IN/OUT/INOUT parameters.

- **IN:** INPUT.
- **OUT:** OUTPUT.
- **IN OUT:** INPUT and OUTPUT.

```
CREATE [OR REPLACE] FUNCTION [SCHEMA..] function_name
    [ (parameter [,parameter]) ]
    RETURN return_datatype
IS | AS
    [declaration_section
        variable declarations;
        constant declarations;
    ]
BEGIN
    [executable_section
        PL/SQL execute/subprogram body
    ]
    [EXCEPTION]
        [exception_section
            PL/SQL Exception block
        ]
END [function_name];
/

DROP FUNCTION function_name;
```

CODE SHEET

Creating a Function

```
CREATE or REPLACE FUNCTION fun1(no in number)
RETURN varchar2
IS
    name varchar2(20);
BEGIN
    select ename into name from emp1 where eno = no;
    return name;
END;
/
```

Calling a Function

```
DECLARE
    no number :=&no;
    name varchar2(20);
BEGIN
    name := fun1(no);
    dbms_output.put_line('Name: ' || ' ' || name);
end;
/
```

Drop a Procedure

```
DROP FUNCTION fun1;
```


Exercises

1. Create a function to display the count of male minions assigned in various missions.
2. Create a function that gets the mission_id and find the corresponding mission status and the estimate.
3. Write a function to receive a minion_id and display its native currency.
4. Find the currency equivalents of INR 100.
5. Find the minions who have done training together.

Week 11

Triggers

A trigger is a PL/SQL block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
    Declaration-statements
BEGIN
    Executable-statements
EXCEPTION
    Exception-handling-statements
END;
```

Advantages

- SQL triggers provide an alternative way to check the integrity of data.
- SQL triggers can catch errors in business logic in the database layer.
- SQL triggers provide an alternative way to run scheduled tasks. By using SQL triggers, you don't have to wait to run the scheduled tasks because the triggers are invoked automatically before or after a change is made to the data in the tables.
- SQL triggers are very useful to audit the changes of data in tables.

Disadvantages

- SQL triggers only can provide an extended validation and they cannot replace all the validations. Some simple validations have to be done in the application layer. For example, you can validate user's inputs in the client side by using JavaScript or in the server side using server-side scripting languages such as JSP, PHP, ASP.NET, Perl, etc.
- SQL triggers are invoked and executed invisible from the client applications, therefore, it is difficult to figure out what happen in the database layer.
- SQL triggers may increase the overhead of the database server.

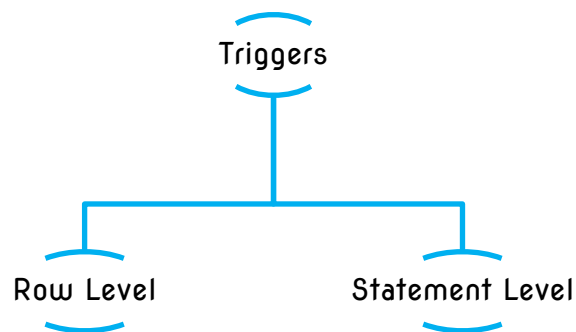


Figure : Types of Triggers

Row Level Triggers

Row Level Trigger is fired each time row is affected by Insert, Update or Delete command. If statement doesn't affect any row, no trigger action happens.

Statement Level Triggers

This kind of trigger fires when a SQL statement affects the rows of the table. The trigger activates and performs its activity irrespective of number of rows affected due to SQL statement.

Types of Triggers based on the time of firing

- ✓ Before Insert Trigger
- ✓ After Inset Trigger
- ✓ Before Update Trigger
- ✓ After Update Trigger
- ✓ Before Delete Trigger
- ✓ After Delete Trigger

Displaying Trigger Errors

If we get a message Warning: Trigger created with compilation errors. You can check the error messages with:

```
Show errors trigger <trigger_name>;
```

Viewing Defined Triggers

To view all the defined triggers, use:

```
select name_of_trigger from user_triggers;
```

For more details on a particular trigger:

```
select trigger_type, triggering_event, table_name,  
referencing_names, trigger_body  
from user_triggers  
where trigger_name = '<name_of_trigger>';
```

Disabling/ Enabling Triggers

```
alter trigger <name_of_trigger> {disable | enable};
```

CODE SHEET

After Update Trigger

```
SET SERVEROUTPUT ON SIZE 1000000

CREATE OR REPLACE TRIGGER CHK_UPDATE
AFTER UPDATE
ON MISSION;
BEGIN
    DBMS_OUTPUT.PUT_LINE(' Update operation on Mission');
END;

/
```

Before Update Trigger

```
CREATE OR REPLACE TRIGGER CHANGE_AGE
BEFORE INSERT
ON MINION;
BEGIN
    UPDATE TABLE SET AGE = AGE +1 WHERE NATIONALITY = 'USA');
END;

/
```

Try to insert values to mission and check the age getting incremented for all minions of nationality 'USA'.

Exercises

1. Create a trigger to increment the level to all the tuples in evilness before any insert on the relation Evilness. Say, if u are inserting an evilness “Burp” for level 2, the existing tuples with level 2, 3 and 4 should be updated as 3, 4 and 5.
2. Create a trigger to alert the user on other training courses offered by the same trainer, when tuples are inserted to relation training.
3. Create a table ‘Currency_Rate_History’ with attributes Currency_id (number), rate (number) – referring to Currency(b_currency) and date (date). Create a trigger that updates the table Currency_Rate_History’ before any update to the values of b_currency in Currency table.
4. Make a trigger to deny update to Payment relation if payment is initiated before the completion of mission.
5. Trigger an Exception as ‘You can’t kill Minions!!!’ on trying to delete tuples from Minion.
6. Create a table identical to takes_training as Takes_Training_History Create a trigger that updates a row in ‘Takes_Training_History’ when any tuples in takes_training gets deleted.

DATASHEET

Minions (19+1)

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

Master (10)

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

Language (10)

English
French
Spanish
Chinese
Urdu
German
.
.
.
.
.

Skills (10)

Sneeze
Itch
Ticklish
Yawn
.
.
.
.
.

Currency(5)

INR
USD
EUR
GBP
JPY

Country (10)

USA
Germany
Japan
France
India
.
.
.
.
.

Mission: Hide pages hereafter

Minion Hired: Kevin

Evil Master: Zugi

Duration: Until next week



You have successfully completed Week 11. Come back next week!!