TRIGGER A block of SQL code that automati--cally gets executed before or after an insert, update or delete argument

```
DROP TRIGGER IF EXISTS paymentsAfterInsert
DELIMITER $$            table name _after/before_action
                                ↓
CREATE TRIGGER paymentsAfterInsert
    AFTER INSERT ON payments
    FOR EACH ROW
BEGIN
    UPDATE invoices
    SET payment_total = payment_total
                      + NEW.amount
    WHERE invoice_id = NEW.invoice_id;
END $$
DELIMITER ;
```

- `SHOW TRIGGERS;` : To veiw all triggers

  `SHOW TRIGGERS LIKE 'payments%'`

- `DROP TRIGGER IF EXSTS triggername ;`

Date
Page No. ✓

Events A task (or block of SQL code) that gets executed according to a schedule

SHOW VARIABLES LIKE 'event%';

```
DROP EVENTS IF EXISTS yearly-delete_st
DELIMITER $$
CREATE EVENTS yearly_delete_stale audit_rows
ON SCHEDULE
    EVERY 1 YEAR STARTS '2021-01-01'
                 ENDS '2029-01-01'
DO BEGIN
    DELETE FROM payments_audit
    WHERE action_date < NOW()-INTERVA
END $$
DELIMITER ;
```

- SHOW EVENTS;
  SHOW EVENTS LIKE 'yearly%';
- DROP EVENTS IFEXISTS name-of-events
- ALTER EVENT - To alter the event.
  - use at the place of CREATE EVEN
  - To disable or enable

ALTER EVENT __________ DISABLE;
ALTER EVENT __________ ENABLE;

Date
Page No.

Transaction Group of SQL statement that represent a single unit of work.

_audit_rows;

$$\begin{tikzcd} & \text{ACID Properties} \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\ \text{Atomicity} & \text{Consistency} & \text{Isolation} & \text{Durability} \end{tikzcd}$$

```
START TRANSACTION;
:
COMMIT; or R
START TRANSACTION;
:
ROLL BACK;
```

1YEAR;

	Lost Updates	Dirty Read	Non-repeating read	Phantom Reads
Read uncommitted				
Read committed		✓		
Repeatable read	✓	✓	✓	
Serializable	✓	✓	✓	✓

```
SET TRANSACTION ISOLATION LEVEL name;
SET SESSION
SET GLOBAL
```

Date
Page No. ✓

Deadlock

My SQL Data Types

* STRINGS

- CHAR(x) - fixed length
- VARCHAR(x) - variable length
  (Eg. Password, Username etc.
  max - 65,535 characters ≈ 64KB
- MEDIUMTEXT - 16 MB (max)
- LONGTEXT - 4 GB (max)
- TINYTEXT - 225 bytes (max)
- TEXT - 64 KB (max)

* INTEGERS

- TINYINT 1b [-128, 127]
- UNSIGNED TINYINT [0, 255]
- SMALLINT 2b [-32K, 32K]
- MEDIUMINT 3b [-8M, 8M]
- INT 4b [-2B, 2B]
- BIG INT 8b [-9Z, 9Z)

* ZEROFILL

Eg. INT (4) ⇒ 0001

Date
Page No.

* FLOATING POINT

- DECIMAL (p,s) - DEC, NUMERIC, FIXED
- DOUBLE 8b
- FLOAT 4b

* BOOLEANS TYPE

- BOOL BOOLEAN

* ENUMS AND SETS

- ENUM ('small', 'medium', 'large')
  don't use much
- SET(.....)
  don't use much

* DATE / TIME

- DATE
- TIME
- DATETIME 8b
- TIMESTAMP 4b (up to 2038)
- YEAR

* BLOBS

- BLOB 65KB
- TINYBLOB 255b
- MEDIUMBLOB 16MB
- LONGBLOB 4GB

* JSON Lightweight format for storing and transferring data over the Internet

```
JSON
{
  "key" : Value
}
```

```
UPDATE products
SET properties = '
{
  "dimensions" : [1, 2, 3],
   "weight" : 10,
  ,"manufactures" : {"name" : "Sony"}
}'
WHERE product_id = 1;
```

OR

```
UPDATE products
SET properties= JSON_OBJECT
  ('weight', 10,
   'dimension', JSON_ARRAY(1,2,3),
   'manufacturing', JSON_OBJECT('name','Sony')
  )
 WHERE product-id = 1;
```

```
SELECT product-id, JSON_EXTRACT (properties, '$.weight')
FROM products
WHERE productid = 1;
```

Date
Page No.

~~OR~~
~~SELECT product id, properties -> '$.di~~
~~ER~~SELECT
properties -> '$. dimensions[0]'
properties -> '$. manufacture.name'
to remove quote from result
->>

JSON- REMOVE
(properties,
'$.age')

```
UPDATE products
SET properties = JSON- SET(
        properties,
        '$.weight', 20
        '$.age', 10 )
  WHERE product_id = 1;
```

weight')

Date			
Page No.			

Creating Database

Data Modelling (i) Understand the requirements
(ii) Build a conceptual model
(iii) Build a logical Model
(iv) Build a physical Model

ER — Entity Relationship

UML — Unified modeling language

Microsoft Visio
draw.io
LucidCharts

Normalization

1NF: Each cell should have single value and we can't have repeated coloumn

2NF: Every table should describe one entity and every column in that table ~~s~~ should describe that entity

3NF: A column in table should not be drive from other column

```
INSERT INTO "table_name" ("col1", "col2", ...)
VALUES ("Value1", "Value 2", ...)
```

create database

```
CREATE DATABASE IF NOT EXISTS
  name1 ;
DROP DATABASE IF EXISTS name;
```

create Table

```
~~DRO~~ DROP TABLE IF EXISTS name ;
CREATE TABLE name
 (
   customer_id INT PRIMARY KEY
                 AUTO_INCREMENT,
   name      VARCHAR(50) NOT NULL,
   points    INT NOT NULL DEFAULT 0,
   email     VARCHAR NOT NULL UNIQUE
 );

CREATE TABLE IF NOT EXISTS name

DROP TABLE IF EXISTS name ;
CREATE TABLE name
```

Alter Table

```
 ALTER TABLE
     ADD last_name VARCHAR(50) NOT NULL
     AFTER first_name,
     MODIFY age VARCHAR(55),
     DROP points ;
```

Creating Relationship ⇒

```
CREATE TABLE orders
( order_id INT PRIMARY KEY,              customer_id
  customer_id INT NOT NULL,
  FOREIGN KEY fk_orders_customers(^ )
    REFERENCES customers (customer_id)
     ON UPDATE CASCADE
     ON DELETE NO ACTION );
```

ANALYZE TABLE table_name ;

create Index CREATE INDEX ind_name ON table-name (column.name);

SHOW INDEXS IN table_name;

For VARCHAR and CHAR we can also give limit for character.
last_name (20)

Full Text Indexe CREATE FULLTEXT INDEX idx.name ON table-name (column, column);

Composite Indexes CREATE INDEX idx.name ON table-name (column, column);

* Put the most frequently used column first
Put the column with highest cardinality first (Take your queries in account)

Date			
Page No.			

```
COPY "table_name" ("column1", "column2")
FROM 'C:\tmp\persons.csv'
        DELIMITER ',' CSVHEADER;
```

```
ALTERTABLE "table-name"
• ADD "column_name" "Data-type";
• DROP COLUMN
• DROP "column_name"
• ALTER COLUMN "column_name" TYPE "N Data_Type";
• RENAME COLUMN "column1" TO "Column2";
• ALTER COLUMN "column1" SET NOT NULL;
• ALTER COLUMN "column1" DROP NOT NULL;
• ADD CONSTRAINT "col-name" CHECK ("col-name" >=100);
• ADD PRIMARY KEY ("column_name");
• ADD CONSTRAINT "child_col" FOREIGN KEY ("parent_col")
  REFERENCES "parent-table";
```

EXCEPT It is used to return all rows in the first SELECT statement that are not returned by the second SELECT statement

```
SELECT  column1, column2, ...
FROM    table 1
WHERE   conditions
EXCEPT
SELECT  exp1, exp2, ...
FROM    table2.
WHERE   conditions
```

Date
Page No.

STRING AGGREGATOR £
STRING_AGG (expression, delimeter)
aggregate every value in column

POWER (6,2) 6^2

AGE ([date1,] date2)
↳ if not mentioned, current date will be used.

TO_CHAR (value, format_mask)

To_char (order_date, 'MMDDYY')

SETSEED (seed)

Seed can have a value between 1.0 and -1.0 inclusive.

```
SELECT SETSEED(0.5);
SELECT RANDOM();
SELECT RANDOM();
```

EXTRACT ('unit' from 'date')
'day
decade
doy (day of the year)
epoch (No. of sec since 1970-01-01)
hour, minute, year, second

TO_date (string, format-mask)
TO_number (string, format_mask

SOFT DELETE : UPDATE
HARD DELETE : TRUNCATE

Date			
Page No.			

CREATE USER user_name
- [WITH PASSWORD 'password- value']
- VALID UNTIL 'expiration']; ↗(table_name)
- GRANT privileges ON object TO user_name;
- {REVOKE privileges ON object FROM user_name;}

↙
ALL, SELECT, INSERT, UPDATE, DELETE... etc

DROP USER user_name

ALTER USER starteach
RENAME TO ST;

SELECT usename
FROM table_name

SELECT DISTINCT usename
↓
show login user

SELECT DISTINCT *
FROM table_name (to see activity)

CREATE TABLESPACE <tablespace name>
LOCATION < location on drive>

EXPLAIN
VACCUM [table_name]
Delete all soft delete

CREATE SCHEMA schema_name;