Database
Week #2

Database Keys

Primary Key

Foreign Key

Unique Key

Compound Key

PRIMARY KEY

is a key in a relational database that is unique for each record.

- Primary key cannot have a NULL value.
- Each table can have only one primary key.
- Primary key can be related to another tables as a Foreign Key.
- Primary key supports Auto Increment value.
- We can't delete primary key value from the parent table which is used as a foreign key in child table. To delete we first need to delete that primary key value from the child table.

PRIMARY KEY EXAMPLE

```
create table product(
    id int not null,
    name varchar(100),
    description varchar(100),
    harga float,
    primary key (id)
);
```

FOREIGN KEY

is a field (or collection of fields) in one table, that refers to the Primary Key in another table.

- Foreign key can accept multiple null value.
- Foreign key do not automatically create an index, clustered or non-clustered. You can manually create an index on foreign key.
- We can have more than one foreign key in a table.
- We can delete the foreign key value from the child table even though that refers to the primary key of the parent table.

FOREIGN KEY EXAMPLE

```
create table wishlist(
    id int not null auto_increment,
    id_product varchar(10) not null,
    description text,
    primary key (id),
    constraint fk_wishlist_product
    foreign key (id_product) references product (id)
);
```

UNIQUE KEY

A unique key is a group of one or more than one fields or columns of a table which uniquely identify database record.

- Unique Constraint may have a NULL value.
- Each table can have more than one Unique Constraint.
- By default, Unique key is a unique non-clustered index.
- Unique Constraint can not be related with another table's as a Foreign Key.

UNIQUE KEY EXAMPLE

```
create table customer(
    id int not null auto_increment,
    email varchar(100) not null,
    first_name varchar(100) not null,
    last_name varchar(100),
    primary key (id),
    unique key email_unique (email)
);
```

COMPOUND KEY

is a key that consists of 2 or more attributes that uniquely identify an entity occurrence.

- Compound key cannot have a NULL value.
- A compound key, in the context of relational databases, is a combination of two or more columns in a table that can be used to uniquely identify each row in the table. Uniqueness is only guaranteed when the columns are combined; when taken individually the columns do not guarantee uniqueness.

COMPOUND KEY EXAMPLE

```
create table product(
    id int not null,
    nama varchar(100),
    manufacturer varchar(100),
    primary key (nama, manufacturer)
);
```

CONSTRAINT

- Constraints 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 table.
- If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be column level or table level.

CONSTRAINT EXAMPLE

```
alter table product
add constraint price_constraint check (price > 10000);

-- check if the constraint is activated
insert into product (id, name, price, quantity)
values ('P0006', 'Mie Goreng Spesial', 9000, 50 );
#will fail because the price is < 10000

insert into product (id, name, price, quantity)
values ('P0006', 'Mie Goreng Spesial', 11000, 150 );
#will work because the price is > 10000
```

Different Types of SQL JOINs

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
-

JOINs EXAMPLE

-- INNER JOIN --

```
# Using MySQL Sample Database : Sakila

select c.first_name as 'customer_name',
st.first_name as 'staff_name'
from customer c
inner join store s on s.store_id = c.store_id
inner join staff st on st.store_id = s.store_id
group by c.first_name
limit 10;
```

JOINs EXAMPLE (2)

-- LEFT JOIN --

#Using MySQL Sample Database : Sakila

```
select f.title as 'film_title', f.release_year, l.name as
'language_name'
from film f
left join `language` l on l.language_id = f.language_id
group by f.film_id
limit 15;
```

JOINs EXAMPLE (3)

-- RIGHT JOIN --

#Using MySQL Sample Database : Sakila

```
select f.title as 'film_title', f.release_year, l.name as 'language_name'
from film f
right join `language` l on l.language_id = f.language_id
group by f.film_id
order by 1
limit 15;
```

UNION & UNION ALL

- The MySQL UNION operator is used to combine the result sets of 2 or more SELECT statements

- UNION ALL keeps all of the records from each of the original data sets, while UNION removes any duplicate records

UNION / UNION ALL EXAMPLE

```
#Using MySQL Sample Database : Sakila

(select distinct first_name as 'Name', 'customer' as 'identity'
from customer limit 10)

UNION     # or UNION ALL

(select distinct title as 'Name', 'title' as 'identity'
from film limit 10);

#UNION removes any duplicate records
```

INDEX & VIEW

- Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

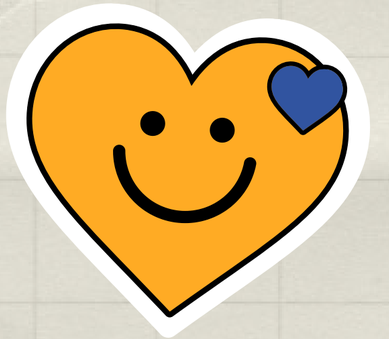
INDEX EXAMPLE

```
create table sellers(
    id int not null auto_increment,
    name varchar(100),
    email varchar(100),
    primary key (id),
    unique key email_unique(email),
    index(name)
);
```

VIEW EXAMPLE

```
create view film_join_category_view as (
    select f.film_id, f.title, f.release_year, c.category_id, c.name
    from film f
    inner join film_category fc on fc.film_id = f.film_id
    inner join category c on c.category_id = fc.category_id
    limit 10
);
```

THANK YOU!



DATE

S M T W T F S

- Team 3 -
Data Scientist A