JIGSAW ACADEMY

THE ONLINE SCHOOL OF ANALYTICS

SQL

Roadmap

1. **INTRODUCTION TO SQL**

   1.1. What is SQL and RDBMS?

   1.2. How does RDBMS work?

   1.3. Normalization

   1.4. Types of databases

   1.5 SQL terminology

   1.6 List of basic commands

What is SQL and RDBMS?

Structured **Q**uery **L**anguage

"Sequel"

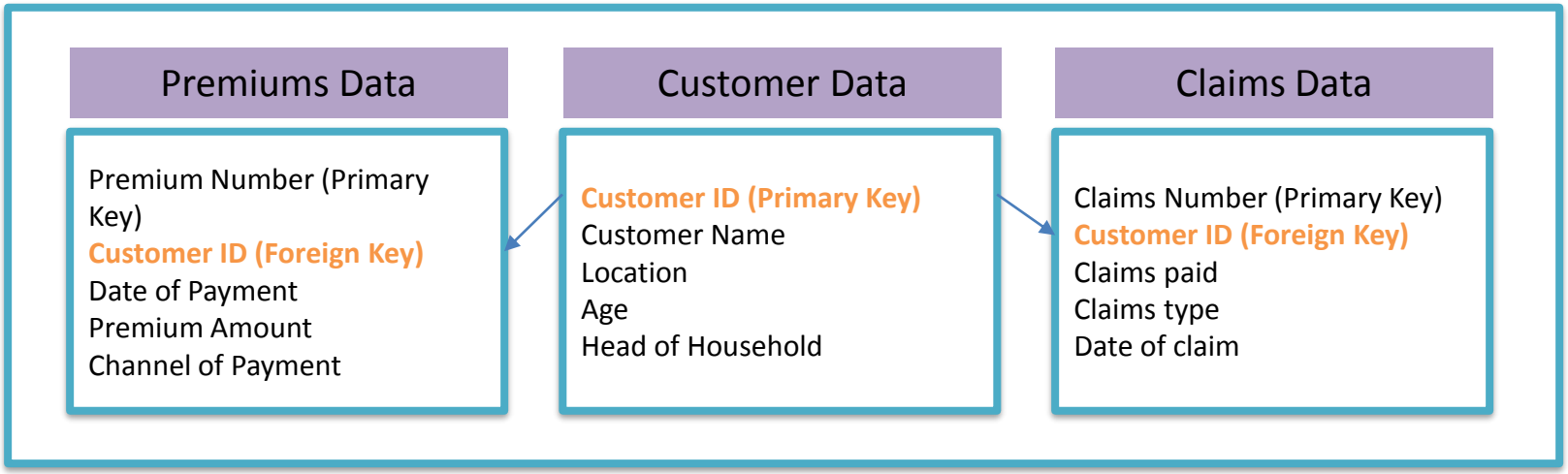
SQL is a standard query language for the definition, manipulation and control of **relational databases**

Special system software in which all data is stored in relations to each other in tables. Ex: Oracle, IBM DB2

The organization of data in relational databases is based on relational model proposed by E. F. Codd in 1970. It is different from other database systems like hierarchical databases and network databases

How does RDBMS work?

EXAMPLE OF RDBMS IN INSURANCE:



Normalization

Why normalize?

Organizing data in an efficient way in a database is called normalization:

1. To remove data redundancy
2. To ensure data dependencies make sense

How to normalize?

NORMALIZATION GUIDELINES

1NF

- Define data terms
- No groups of data in columns
- Ensure primary key

2NF

- Ensure 1 NF compliance
- Ensure there are no partial dependencies

3NF

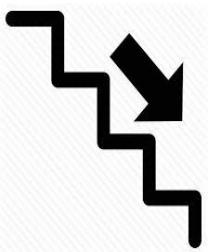
- Ensure 2 NF compliance
- All non primary fields are dependent on the primary key/ No transitive dependency

1NF

1NF

- Define data terms
- No groups of data in columns
- Ensure primary key

1 Define data terms



Identify the data to be stored

Create columns and column types from the data

Group the columns to a table/tables

2 No groups of data in columns

Before

ID	Name	Age	Item
100	Ted	25	Apples, Oranges
200	Ned	30	Grapes, Apples

After

ID	Name	Age	Item
100	Ted	25	Apples
100	Ted	25	Oranges
200	Ned	30	Grapes
200	Ned	30	Apples

3 Ensure primary key

Primary key uniquely identifies each record in a table. It contains unique values and can be made up of single or multiple variables.

2NF

2NF

- Ensure 1 NF compliance
- Ensure there are no partial dependencies

No partial dependencies

Before

ID	Name	Age	Item
100	Ted	25	Apples
100	Ted	25	Oranges
200	Ned	30	Grapes
200	Ned	30	Apples

After

ID	Item
100	Apples
100	Oranges
200	Grapes
200	Apples

ID	Name	Age
100	Ted	25
200	Ned	30

3NF

3NF

- Ensure 2 NF compliance
- All non primary fields are dependent on the primary key/ No transitive dependency

No transitive dependencies

Before

ID	Name	Age	Zip code	Place
100	Ted	25	22	ABC
200	Ned	30	22	ABC
300	Led	35	33	XYZ

After

ID	Name	Age	Zip code
100	Ted	25	22
200	Ned	30	22
300	Led	35	33

Zip code	Place
22	ABC

RDBMS databases

RDBMS Name	Owned by	Creation year	Open Source?	Commercial licensing?	Used by	Other information
MySQL	MySQL AB	1995	Yes	Yes	Both large and small databases. Internet databases like Wikipedia, Moodle etc	- One of the fastest databases - Built in security
Microsoft Access	Microsoft	1992	No	Yes	Small companies with less than 100 users	- Can be analysed with excel and supports macros and VBA - Cannot be split over multiple hard drives
DB2	IBM	1983	No	Yes	Majorly by large scale organizations	- Second by market share next only to Oracle - On of the editions has business intelligence capabilities like online analytics
Oracle	Oracle	1979	No	Yes	Majorly by large scale organizations	- Largest by market share - Relatively expensive when compared to DB2
SQL Server	Microsoft	1989	No	Yes	Medium to large scale organizations	- Includes ETL and OLAP functionality - SQL Server 2005 is a lighter version with lower cost

SQL Terminology

SAS Term	SQL Term
Dataset	Table
Observation	Row/Record
Variable	Column/Field
Merge	Join
Missing	Null

List of commands

Data Definition Language (DDL) commands

Create, Alter, Drop, Add/Select, Rename, Modify

Data Control Language (DCL) commands

Grant, Revoke

Data Manipulation Language (DML) commands and techniques

Manipulations - I

Select, Insert, Update, Create View, Case When, Aggregate functions, Scalar functions, Distinct, Duplicates, missing values, outliers

Manipulations - II

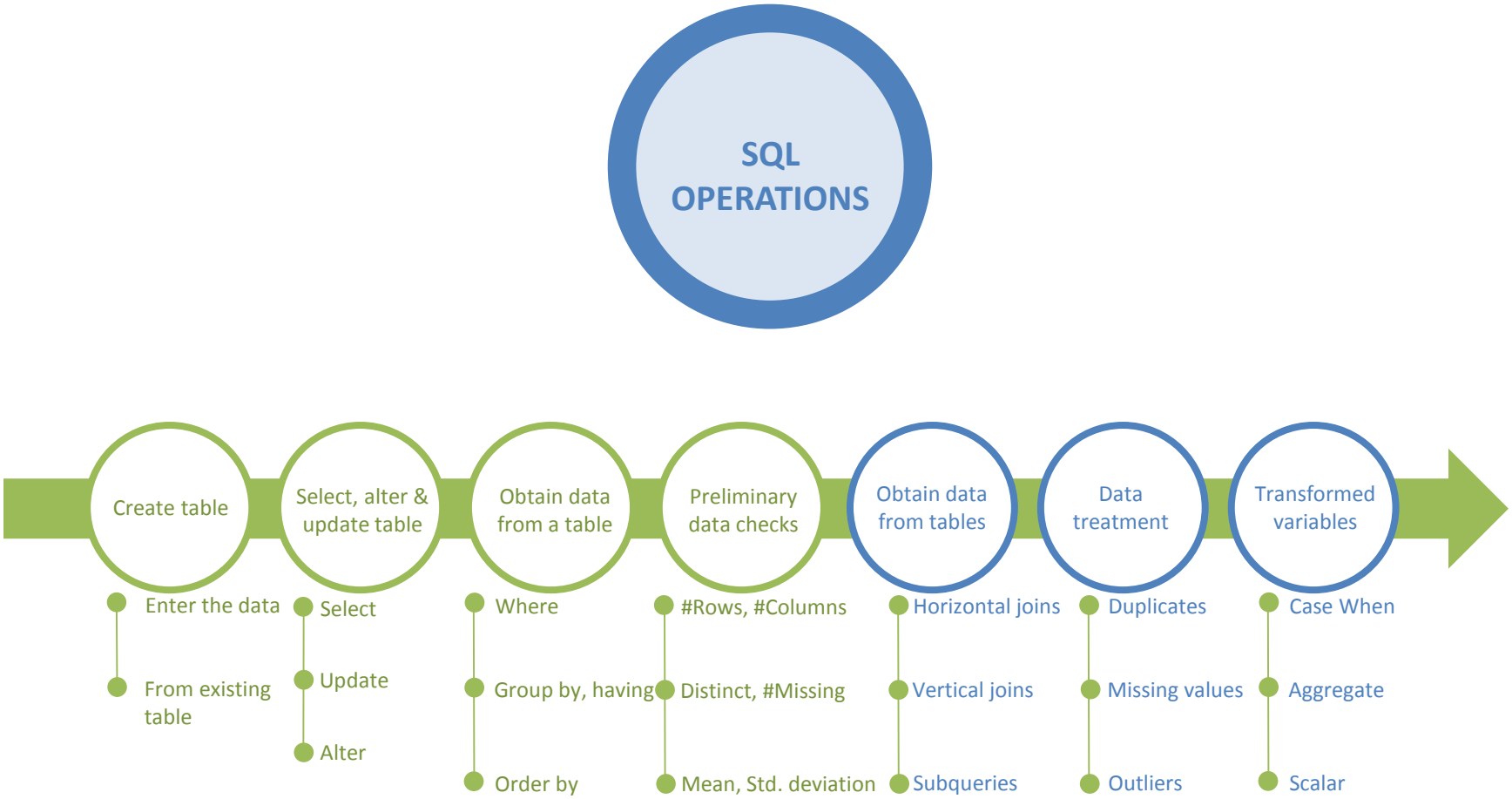
Subqueries, Horizontal joins, Vertical joins

JIGSAW ACADEMY

THE ONLINE SCHOOL OF ANALYTICS

SQL

SQL operations



Create table through manual entry of the data (1/2)

Create table

Enter the data

From existing table

VALUES CLAUSE

1 -- create a table

```
CREATE TABLE table1
(var1 char(2),
var2 int,
var3 date)
;

INSERT INTO table1 (var1, var2, var3)
VALUES
('ab', 41, "2000-03-20"),
('cd', 42, "1987-02-22")
;
```

- SQL, like SAS, is not case sensitive
- -- or /*comment here*/ are used to comment a MySQL query
- All queries are MySQL queries

Create table through manual entry of the data (2/2)

Create table

Enter the data

From existing table

SET CLAUSE

2

```
CREATE TABLE table2
(var1 char(2),
var2 int,
var3 date)
;

INSERT INTO table2
SET
Var1 = 'ab', var2 = 41, var3 = "2000-03-20" ;
```

Create table from existing table (1/3)

Create table

Enter the data

From existing table

SELECT STATEMENT

1

```
CREATE TABLE table3 AS
SELECT var1, var2, var3
FROM table1
;
```

Create table from existing table (2/3)

Create table

Enter the data

From existing table

INSERT INTO STATEMENT

2

```
INSERT INTO table3 (var1, var2, var3)
SELECT var1, var2, var3
FROM table1
;
```

Create table from existing table (3/3)

Create table

Enter the data

From existing table

LIKE CLAUSE

3

```
CREATE TABLE table4 LIKE table1;
INSERT INTO table4 (var1, var2)
SELECT var1, var2
FROM table1
;
```

4

```
CREATE TABLE table5
SELECT * FROM table1
;
```

Select all data from a table

Select, alter & update table

- Select
- Alter
- Update

SELECT * STATEMENT

1

```
SELECT *
FROM table1
;
```

- SELECT TABLE only prints the output and it does not create a table

Select a specified number of records from a table

Select, alter & update table

- Select
- Alter
- Update

SELECT STATEMENT WITH LIMIT

2

```
SELECT *
FROM table1
LIMIT 1
;
```

Select the specified columns from a table

Select, alter & update table

- Select
- Alter
- Update

SELECT STATEMENT WITH VARIABLE NAMES

3

```
SELECT var1, var2
FROM table1
;
```

Alter table by adding new columns (1/4)

Select, alter & update table

- Select
- Alter
- Update

ALTER TABLE WITH ADD CLAUSE

1

```
ALTER TABLE table1
ADD
(var4 int,
var5 char(2))
;
```

- Adds empty columns into the table. Update to add data to these columns

Alter table by dropping columns (2/4)

Select, alter & update table

- Select
- Alter
- Update

ALTER TABLE WITH DROP CLAUSE

2

```
ALTER TABLE table1
DROP var4,
DROP var5
;
```

Alter table by modifying columns (3/4)

Select, alter & update table

- Select
- Alter
- Update

ALTER TABLE WITH MODIFY CLAUSE

3

```
ALTER TABLE table1
MODIFY COLUMN Var1 char(60),
MODIFY COLUMN Var2 char
;
```

Alter table by renaming columns (4/4)

Select, alter & update table

- Select
- Alter
- Update

ALTER TABLE WITH CHANGE CLAUSE

4

```
ALTER TABLE table1
CHANGE COLUMN Var1 customer_name char(60),
CHANGE COLUMN Var2 customer_ID char(20)
;
```

Difference between Insert into and Update statements

Insert into		Update
Insert Into does not rewrite on existing data	REWRITE	Update table rewrites on exiting data
Insert Into adds entirely new rows	NEW ROWS	Update does not add entirely new rows
Total rows increase in the dataset as a result of Insert Into	TOTAL ROWS	Total rows do not increase as a result of Update
Insert into cannot be used on existing rows	EXISTING ROWS	Update can be used on existing rows of data only

Updating records within a table

Select, alter & update table

- Select
- Alter
- Update

UPDATE TABLE WITH SET CLAUSE (WITHIN TABLE)

1

```
UPDATE table1
SET
var2 = var2*2
;
```

- In My SQL Go to Edit > Preferences > SQL Editor > Uncheck Safe updates and reconnect for the above command to work

Updating records from another table

Select, alter & update table

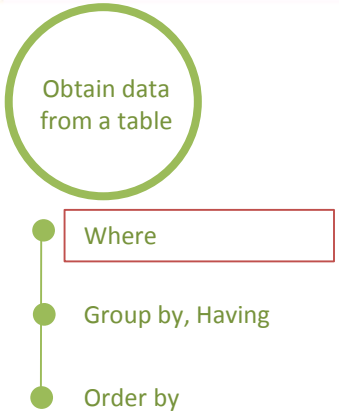
- Select
- Alter
- Update

UPDATE TABLE WITH SET CLAUSE (OTHER TABLES)

2

```
UPDATE table1 as a
SET var4 = (SELECT var2 FROM table2 as b
            where a.var1 =b.var1);
```

Where condition to select rows from a table



SELECT STATEMENT - WHERE CLAUSE

1

```
SELECT var1, var2, avg(var3) as A from table1
WHERE var1 > 87
;
```

Difference between IS NOT NULL and <> NULL

```
SELECT var1, var2, avg(var3) as A from table1
WHERE var1 IS NOT NULL
;
```

```
SELECT var1, var2, avg(var3) as A from table1
WHERE var1 <> NULL
;
```

- Conditional operators are = (equal to), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), !=(not equal to)
- Logical operators are AND (&), OR (|)
- You cannot perform any arithmetic operations on NULL
- In MySQL, 0 OR NULL means FALSE and everything else is TRUE
- If you perform logical operation IS NOT NULL it is same as IS NOT zero/NULL. If you perform <> NULL then it is same as <> NULL only and not <> NULL/zero

Group by clause to roll-up the data (1/2)

Obtain data from a table

- Where
- Group by, Having
- Order by

SELECT STATEMENT – GROUP BY CLAUSE

2

```
SELECT var1, var2, avg(var3) as A from table1
WHERE var1 > 87
GROUP BY var1, var2
;
```

- Group by rolls-up (aggregates) the data at a level specified by the group by-variables (variables specified in group by statement – var1 and var2)
- All unique combinations of data from the group by-variables are tabulated
- Then, for each of these unique combinations of group by-variables the aggregated results are displayed for the aggregated variable – var3

Group by clause to roll-up the data (2/2)

Obtain data from a table

- Where
- Group by, Having
- Order by

I/P table

Student	Var1 (Studies)	Var2 (Games)	Var3 (age)
1	R	R	4
2	G	R	4.5
3	R	R	3.5
4	G	G	4.5
5	G	G	4.5
6	G	R	4

Query

```
SELECT var1,
var2, avg(var3)
as A
from table1
GROUP BY
var1, var2
;
```

O/P table

Var1 (Studies)	Var2 (Games)	Var3 (age)
R	R	3.75
G	R	4.5
G	G	4.5
G	R	4

Having clause to apply a condition on rolled-up data (1/3)

Obtain data from a table

- Where
- Group by, Having
- Order by

SELECT STATEMENT – HAVING CLAUSE

1

```
SELECT var1, var2, avg(var3) as A from table1
GROUP BY var1, var2
HAVING A > 4;
;
```

Having clause to apply a condition on rolled-up data (2/3)

Obtain data from a table

- Where
- Group by, Having
- Order by

Query

```
PROC SQL;

    SELECT var1,
    var2, avg(var3)
    as A from
    table1
    GROUP BY var1,
    var2
    HAVING A >4;

QUIT;
```

I/P table

Student	Var1 (Studies)	Var2 (Games)	Var3 (age)
1	R	R	4
2	G	R	4.5
3	R	R	3.5
4	G	G	4.5
5	G	G	4.5
6	G	R	4

Intermediate table

Var1 (Studies)	Var2 (Games)	Var3 (age)
R	R	3.75
G	R	4.5
G	G	4.5
G	R	4

Having clause to apply a condition on rolled-up data (3/3)

Obtain data from a table

- Where
- Group by, Having
- Order by

SELECT STATEMENT – HAVING CLAUSE

O/P table

Var1 (Studies)	Var2 (Games)	Var3 (age)
G	R	4.5
G	G	4.5

Difference between where and having clause

	Where	Having
DATA	Applied on base data before group by	Applied on aggregated data by group by
POSITION	Before group by in the query	After group by in the query
SELECTION	Cannot be used on aggregated column	Used on aggregated column and preferably not to be used in the absence of aggregated column in the clause
GROUP BY	Is not impacted by group by	Is impacted by group by and when group by is absent having performs like where

Order by clause to sort the data

Obtain data from a table

- Where
- Group by, Having
- Order by

SELECT STATEMENT – ORDER BY CLAUSE

1

```
SELECT var1, var2, avg(var3) as A from table1
GROUP BY var1, var2
HAVING A > 4
ORDER BY A asc, Student_Name desc;
;
```

- Default sorting is ascending, i.e., without specifying asc or desc
- Select statement columns are stored as integers 1, 2, 3… (in the order they appear in Select statement) by SQL. Therefore column names can be substituted by integers in group by and order by clauses

Total number of rows in a table using count(*)

Preliminary data checks

- #Columns, #Rows
- Distinct, #Missing
- Mean, Std. deviation

COUNT(*) FUNCTION

1

```
SELECT COUNT(*)
from table1
;
```

2

```
SELECT COUNT(var1)
 from table1
;
```

- Count(var1) counts non NULL values only. Therefore missing values cannot be counted through count function

Total number of columns in a table from Database

Preliminary data checks

- #Columns, #Rows
- Distinct, #Missing
- Mean, Std. deviation

INFORMATION_SCHEMA DATABASE

3

```
SELECT COUNT(*)
FROM INFORMATION_SCHEMA.COLUMNS – is a table
WHERE table_schema = 'test‘
AND table_name = 'table11‘
;
-- select * from information_schema.tables
-- select * from information_schema.columns
```

- In SQL group of tables are stored under schema and group of schema are stored under a database
- INFORMATION_SCHEMA is the information database, the place that stores information about all the other databases that the MySQL server maintains
- show databases; command will list the databases in MySQL server

Total number of distinct rows in a table using distinct

Preliminary data checks

- #Columns, #Rows
- Distinct, #Missing
- Mean, Std. deviation

DISTINCT FUNCTION

4

```
SELECT DISTINCT(var1)
from table1
;
```

- Distinct(var1) includes NULL value in the output if it is present

Total number of missing values in a column

Preliminary data checks

- #Columns, #Rows
- Distinct, #Missing
- Mean, Std. deviation

COUNT FUNCTION

5

```
SELECT COUNT(*) as total,
COUNT(var2) as nonmissing,
COUNT(*) - COUNT(var2) as missing,
((COUNT(*) - COUNT(var2)) * 100) / COUNT(*)
as 'perc_missing'
from table1
;
```

- Since missing values cannot be counted through the count function we calculate the count of non missing values

Mean, standard deviation, min and max of a column

Preliminary data checks

- #Columns, #Rows
- Distinct, #Missing
- Mean, Std. deviation

AVG FUNCTION

6

```
SELECT AVG(var1)
from table1
;
```

7

```
SELECT STDDEV(var1)
from table1
;
```

8

```
SELECT MIN(var1)
from table1
;
```

9

```
SELECT MAX(var1)
from table1
;
```

Union and Union all clause to horizontally join multiple columns

Obtain data from tables

- Horizontal joins
- Vertical joins
- Subqueries

HORIZONTAL JOIN – UNION & UNION ALL

1

```
SELECT var1, var2 from table1 UNION ALL
SELECT var1, var2 from table2
;
```

2

```
SELECT var1, var2 from table1 UNION
SELECT var1, var2 from table2
;
```

- Result sets from two tables can be horizontally joined using Union & Union All in comparison to Insert Into which inserts records to existing table
- Select statements must have same number of columns
- Variables of different data types can be joined using Union and Union All
- Union removes duplicates in the result whereas Union All shows all the records

Left join to return all rows from left table and matched rows from right table

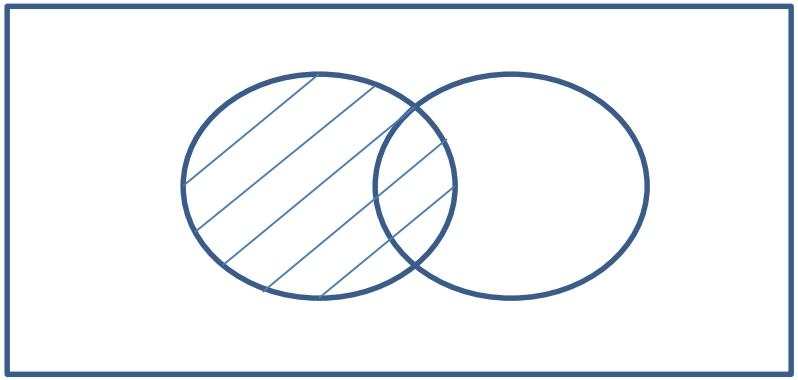
Obtain data from tables

- Horizontal joins
- Vertical joins
- Subqueries

JOINS – LEFT JOIN

1

```
SELECT a.Customer_ID, a.Customer_name,
b.Customer_ID, b.Item_name
FROM
table1 as a
LEFT JOIN
table2 as b
ON a.Customer_ID = b.Customer_ID
;
```



Right join to return all rows from right table and matched rows from left table

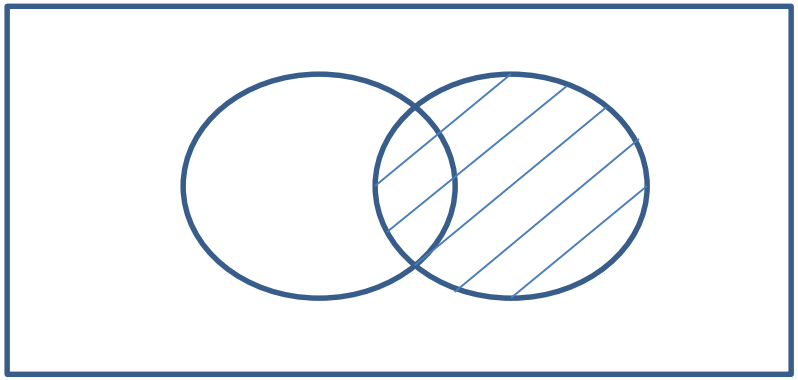
Obtain data from tables

- Horizontal joins
- Vertical joins
- Subqueries

JOINS – RIGHT JOIN

2

```
SELECT a.Customer_ID, a.Customer_name,
b.Customer_ID, b.Item_name
FROM
table1 as a
RIGHT JOIN
table2 as b
ON a.Customer_ID = b.Customer_ID
;
```



Inner join to return matched rows from both the tables

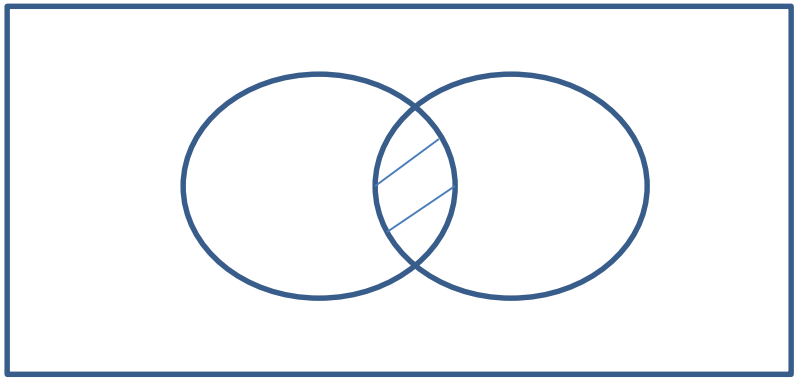
Obtain data from tables

- Horizontal joins
- Vertical joins
- Subqueries

JOINS – INNER JOIN

3

```
SELECT a.Customer_ID, a.Customer_name,
b.Customer_ID, b.Item_name
FROM
table1 as a
INNER JOIN
table2 as b
ON a.Customer_ID = b.Customer_ID
;
```



Indexing with unique column/columns

INDEX – PRIMARY KEY AND UNIQUE

1

```
ALTER TABLE table1
ADD PRIMARY KEY (var1, var3);

ALTER TABLE table1
ADD UNIQUE index_name (var1, var3);

ALTER TABLE table1
DROP PRIMARY KEY ;

ALTER TABLE table1
DROP UNIQUE index_name ;
```

- PRIMARY KEY does not allow columns with NULL values whereas UNIQUE makes this concession
- Values/combination of values in case of 2 columns must be unique in case of primary key or unique index

Indexing with non-unique column/columns

INDEX – ORDINARY INDEX

2

```
ALTER TABLE table1
ADD INDEX index_name (var1, var2);
```

```
ALTER TABLE table1
DROP INDEX index_name ;
```

3

```
SHOW INDEX from table1 ;
```

- Index allows for non-unique values

Nested subqueries execute the inner query first

Obtain data from tables

- Horizontal joins
- Vertical joins
- Subqueries

SUBQUERIES – NESTED

1

```
SELECT * FROM table1
WHERE customer_ID in
(SELECT DISTINCT customer_ID FROM table2)
;
```

- In a nested subquery the inner query is executed first and the output of the inner query is fed to the outer query

Correlated subqueries execute the inner query for each row in outer query, which is executed before the inner query

Obtain data from tables

- Horizontal joins
- Vertical joins
- Subqueries

SUBQUERIES – CORRELATED

2

```
SELECT customer_ID, customer_name
FROM table1 as A
WHERE spend >

(SELECT avg(spend)
FROM table1
WHERE region = A.region)
```

- In a correlated subquery the outer query is executed and for each row of output, the inner query is executed
- Outer table is referenced in the inner query in correlated subqueries

Removal of duplicates using distinct *

Data treatment

- Duplicates
- Missing values
- Outliers

SELECT – DISTINCT *

1

```
CREATE TABLE table_new
SELECT DISTINCT *
FROM table1;
```

Replacement of missing values with some value in a column

Data treatment

- Duplicates
- Missing values
- Outliers

IF NULL FUNCTION

1

```
UPDATE TABLE table1
SET var1 = IFNULL(var2, 45);
```

- Note: This is one of the many ways to treat missing values in data

Creating percentile values for outliers treatment

Data treatment

- Duplicates
- Missing values
- Outliers

@ FUNCTION

1

```
SELECT * from table1
ORDER BY var1;

SET @row:= 0;
SELECT var1, rank/@row as percentile
FROM
(SELECT var1, @row:= @row+1 as rank
FROM table1) as p;
```

Case when function to create bucketed variables

Transformed variables

- Case When
- Coalesce
- Scalar

CASE WHEN FUNCTION

1

```
SELECT var1,

        CASE
        WHEN var2 < 42 THEN 1
        WHEN var2 = 42 THEN 2
        ELSE 3
        END as var4

FROM table1;
```

Coalesce function to return first non null value across variables

Transformed variables

- Case When
- Coalesce
- Scalar

COALESCE FUNCTION

1

```
SELECT *,
COALESCE (jan,feb,mar) as first_income
FROM monthly_income
;
```

Scalar functions for variable transformation

Transformed variables

- Case When
- Coalesce
- Scalar

SCALAR FUNCTIONS

1

```
SELECT
UCASE(var1),
LCASE(var1),
MID(var1,2,2),
LENGTH(var1),
ROUND(var2,1),
NOW() as today
FROM table1
;
```

DCL commands : Grant function to grant privileges

GRANT FUNCTION

1

GRANT privileges on object to user

Ex 1: Grant SELECT on table1 to ‘john’@’localhost’;

Ex 2: Grant SELECT on table1 to *@ ’localhost’;

Ex 3: Grant ALL on table1 to ‘john’@’localhost’;

PRIVILEGES

- SELECT, INSERT, UPDATE, DELETE, INDEX, CREATE, ALTER, DROP, ALL (except grant), GRANT

OBJECTS

- DATABASE NAMES, TABLE NAMES

USER

- User name to which the privileges will be granted to. Type command select current_user(); to obtain your username

DCL commands : Revoke function to revoke privileges

REVOKE FUNCTION

1

REVOKE privileges on object to user

Ex 1: Revoke SELECT on table1 to ‘john’@’localhost’;

Ex 2: Revoke SELECT on table1 to *@ ’localhost’;

Ex 3: Revoke ALL on table1 to ‘john’@’localhost’;

PRIVILEGES

- SELECT, INSERT, UPDATE, DELETE, INDEX, CREATE, ALTER, DROP, ALL (except grant), GRANT

OBJECTS

- DATABASE NAMES, TABLE NAMES

USER

- User name to which the privileges will be granted to