Oracle SQL

Abed Sharifi
Oracle DBA
SQL Developer
Abed.sharifi@gmail.com

What is SQL ?

- **SQL** (**S**tructured **Q**uery **L**anguage)
- **SQL** is a computer language aimed to store, manipulate, and retrieve data in relational databases.
- SQL lets you access and manipulate databases
- SQL is an ANSI (American National Standards Institute) standard

Database concepts

- Database
  - Linking data tables through relationships
  - Controlling data storing and retrieving
  - Maintaining data integrity and accuracy
  - Avoiding data redundancy and inconsistency
  - Sorting and filtering data

Relational Database

- a relational database is a set of tables
- Each table keeps information about one thing

Relational Database Management System (RDBMS)

- is an application that creates, organizes and edits databases
  - Microsoft SQL Server
  - Oracle
  - Microsoft Access

Table Elements

MemberID	LastName	FirstName	Phone	Handicap	JoinDate	Gender
118	McKenzie	Melissa	963270	30	10-May-99	F
138	Stone	Michael	983223	30	13-May-03	M
153	Nolan	Brenda	442649	11	25-Jul-00	F
176	Branch	Helen	589419		18-Nov-05	F
178	Beck	Sarah	226596		06-Jan-04	F
228	Burton	Sandra	244493	26	21-Jun-07	F
235	Cooper	William	722954	14	15-Feb-02	M
239	Spence	Thomas	697720	10	04-Jun-00	M
258	Olson	Barbara	370186	16	11-Jul-07	F

Data Types

Data Type

- System Data Type
- User Defined Data Type

Data Type Categories

- Character Datatypes
- Number Datatype
- DATE Datatype
- LOB Datatypes
- RAW and LONG RAW Datatypes
- ROWID and UROWID Datatypes

https://docs.oracle.com/database/121/SQLRF/sql_elements001.htm#SQLRF0021

Character Datatypes

Data Type	Description
CHAR[(size)]	store fixed-length character data
NCHAR[(size)]	store fixed-length Unicode character data
VARCHAR2(size)	store variable-length character data
NVARCHAR2(size)	store variable-length Unicode character data
Long	store variable-length character data containing up to 2 gigabytes of information

- The LONG datatype is provided for backward compatibility with existing applications. In new applications, use CLOB and NCLOB datatypes for large amounts of character data

Number Datatypes

Data Type	Description
NUMBER [(*p* [, *s*])]	Number having precision *p* and scale *s*. The precision *p* can range from 1 to 38. The scale *s* can range from -84 to 127. Both precision and scale are in decimal digits. A NUMBER value requires from 1 to 22 bytes
FLOAT [(*p*)]	A subtype of the NUMBER data type having precision *p*. A FLOAT value is represented internally as NUMBER. The precision *p* can range from 1 to 126 binary digits. A FLOAT value requires from 1 to 22 bytes.
BINARY_FLOAT	32-bit floating point number. This data type requires 4 bytes
BINARY_DOUBLE	64-bit floating point number. This data type requires 8 bytes

Date Datatypes(1)

Data Type	Description
DATE	Valid date range from January 1, 4712 BC, to December 31, 9999 AD
TIMESTAMP[(*X*)]	Year, month, and day values of date, as well as hour, minute, and second values of time
TIMESTAMP [(*X*)] WITH TIME ZONE	All values of TIMESTAMP as well as time zone displacement value
TIMESTAMP [(*X*)] WITH LOCAL TIM E ZONE	• Data is normalized to the database time zone when it is stored in the database. • When the data is retrieved, users see the data in the session time zone.

X *fractional_seconds_precision*

Date Datatypes(2)

Data Type	Description
INTERVAL YEAR [(*year_precision*)] TO MONTH	Stores a period of time in years and months
INTERVAL DAY [(*day_precision*)] TO SECOND [(*X*)]	Stores a period of time in days, hours, minutes, and seconds

X *fractional_seconds_precision*

LOB Datatypes

Data Type	Description
CLOB	A character large object containing single-byte or multibyte characters
NCLOB	A character large object containing Unicode characters
BLOB	A binary large object. Maximum size is (4 gigabytes - 1) * (database block size)
BFILE	Contains a locator to a large binary file stored outside the database

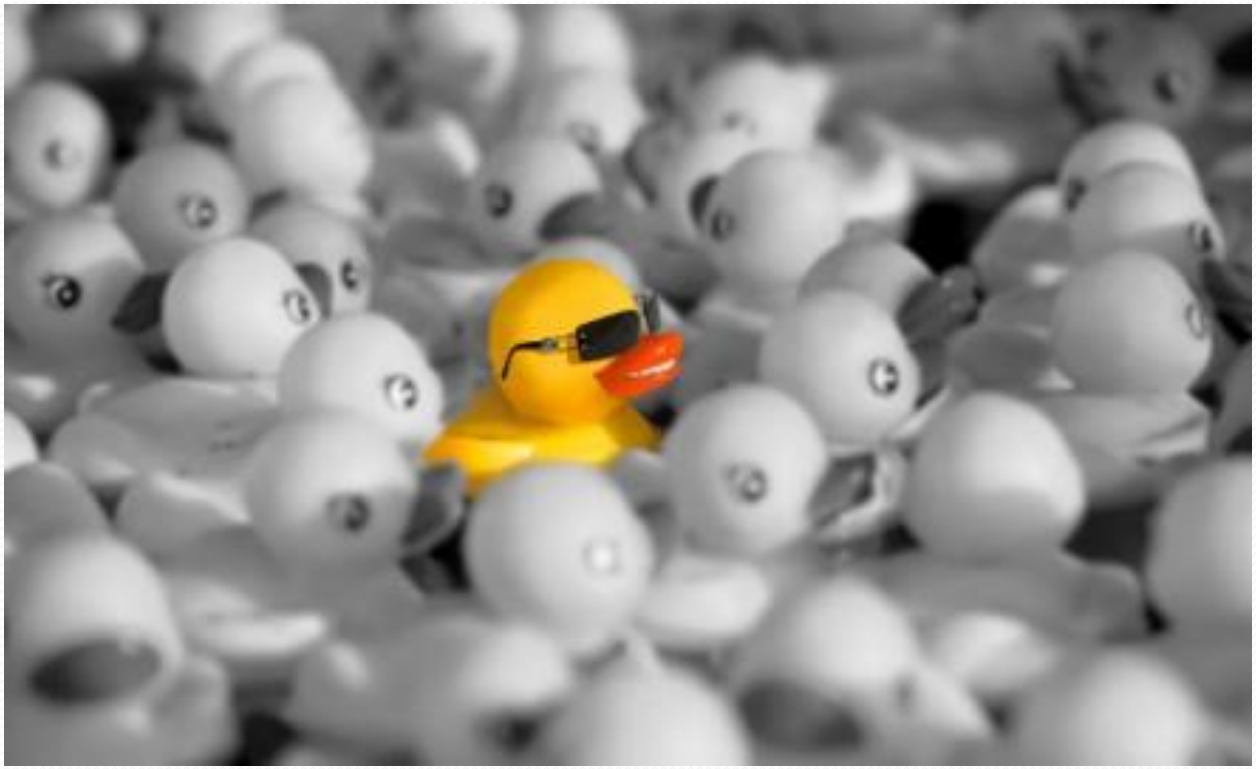
X *fractional_seconds_precision*

Database Constraints

Not null Constraints

- A null value is an unknown
- Null value is not as zero or space.

Unique Constraints



Unique Constraints

- every value in a column or set of columns must be unique.
- When there is not the primary key.

- **Example**
  - no two employees can have
    - the same phone number

Primary key

- The primary key value must be unique and not null.
- Multiple UNIQUE constraints and only one Primary key in a Table .

Foreign key

- FOREIGN KEY in one table points to a PRIMARY KEY in another table.
- prevents that invalid data form being inserted into the foreign key column

Check Constraints

- used to limit the value range that can be placed in a column.

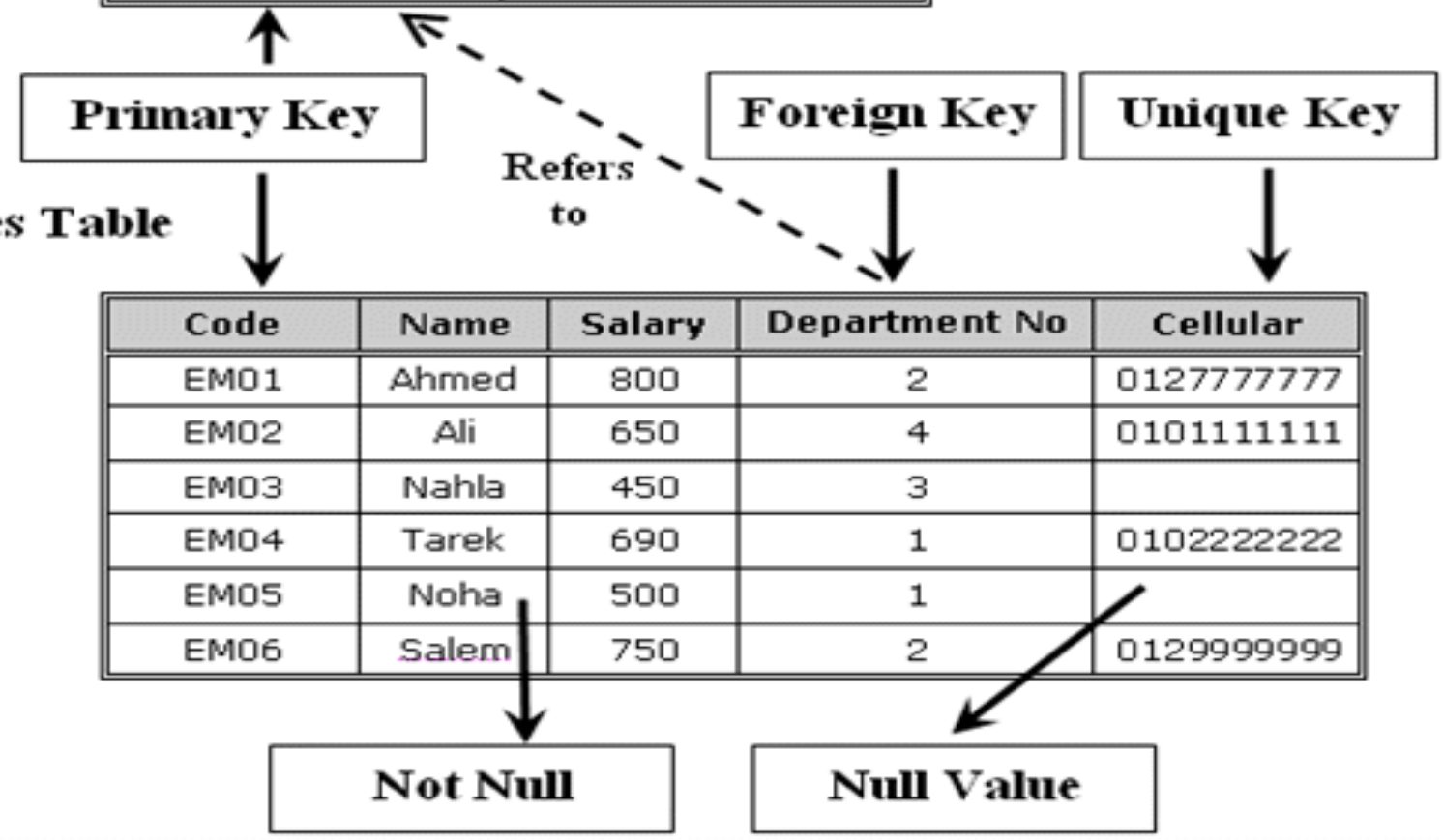
- Example :
  - A column must only include integers greater than 0

DEFAULT Constraint

- used to insert a default value into a column

Departments Table

Department ID	Department Name
1	Training
2	Development
3	Human Resources
4	Sales



Code	Name	Salary	Department No	Cellular
EM01	Ahmed	800	2	0127777777
EM02	Ali	650	4	0101111111
EM03	Nahla	450	3	
EM04	Tarek	690	1	0102222222
EM05	Noha	500	1	
EM06	Salem	750	2	0129999999

Database Relationships

Relationship Types

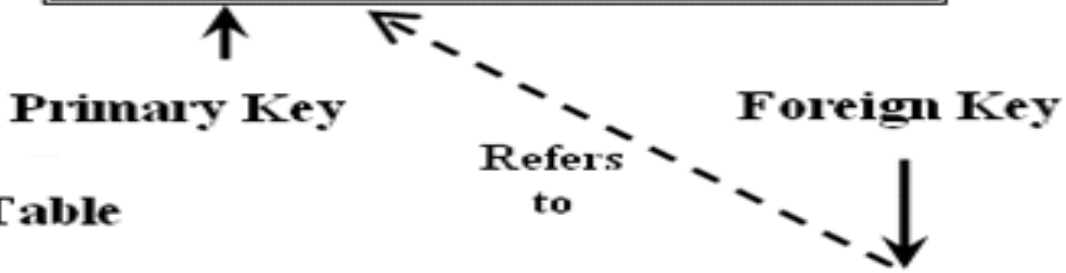
- One-to-one relationship
- One-to-many relationship
- Many-to-many relationship
- Recursive relationship
- Referential integrity

One To One Relationships

One To Many Relationships

Departments Table

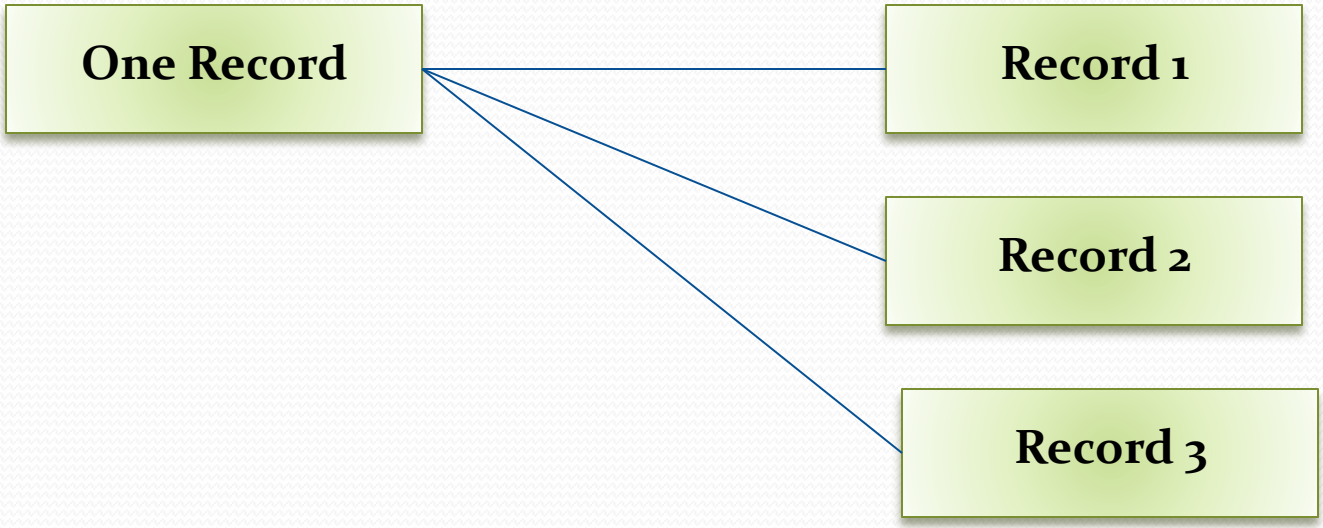
Department ID	Department Name
1	Training
2	Development
3	Human Resources
4	Sales



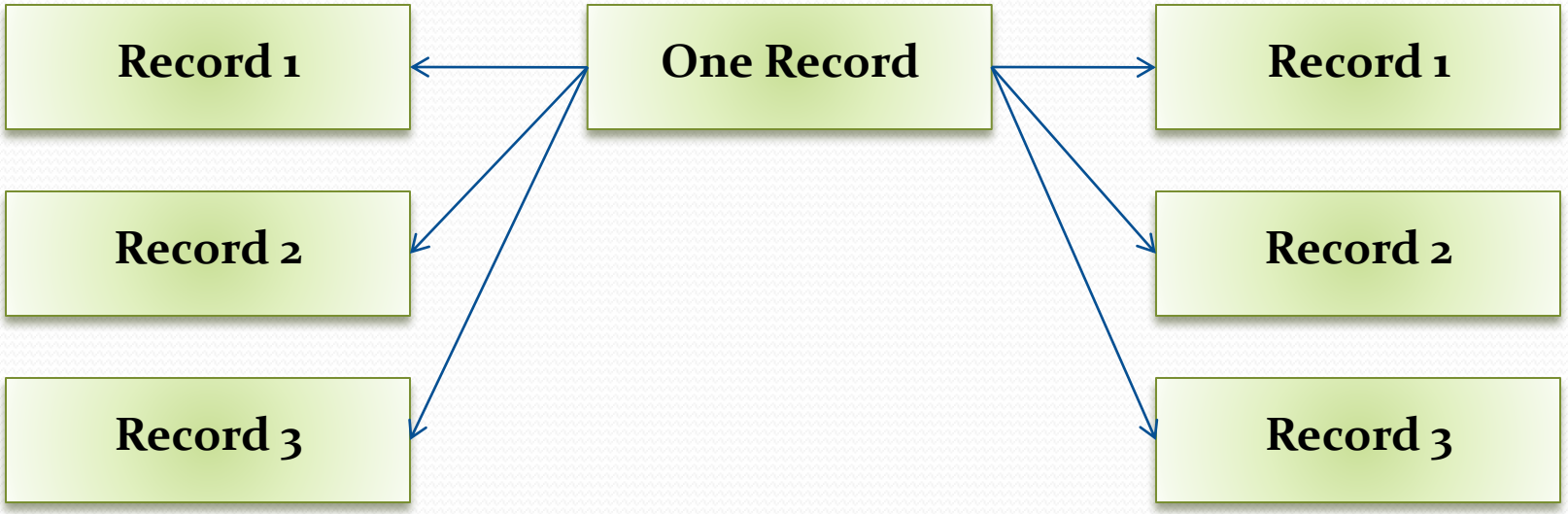
Employees Table

Code	Name	Salary	Department No	Cellular
EM01	Ahmed	800	2	0127777777
EM02	Ali	650	4	0101111111
EM03	Nahla	450	3	
EM04	Tarek	690	1	0102222222
EM05	Noha	500	1	
EM06	Salem	750	2	0129999999

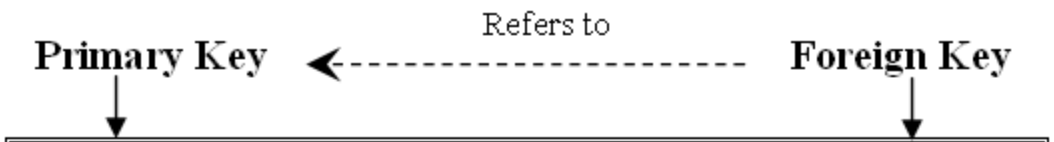
One To Many Relationships



Many To Many Relationships



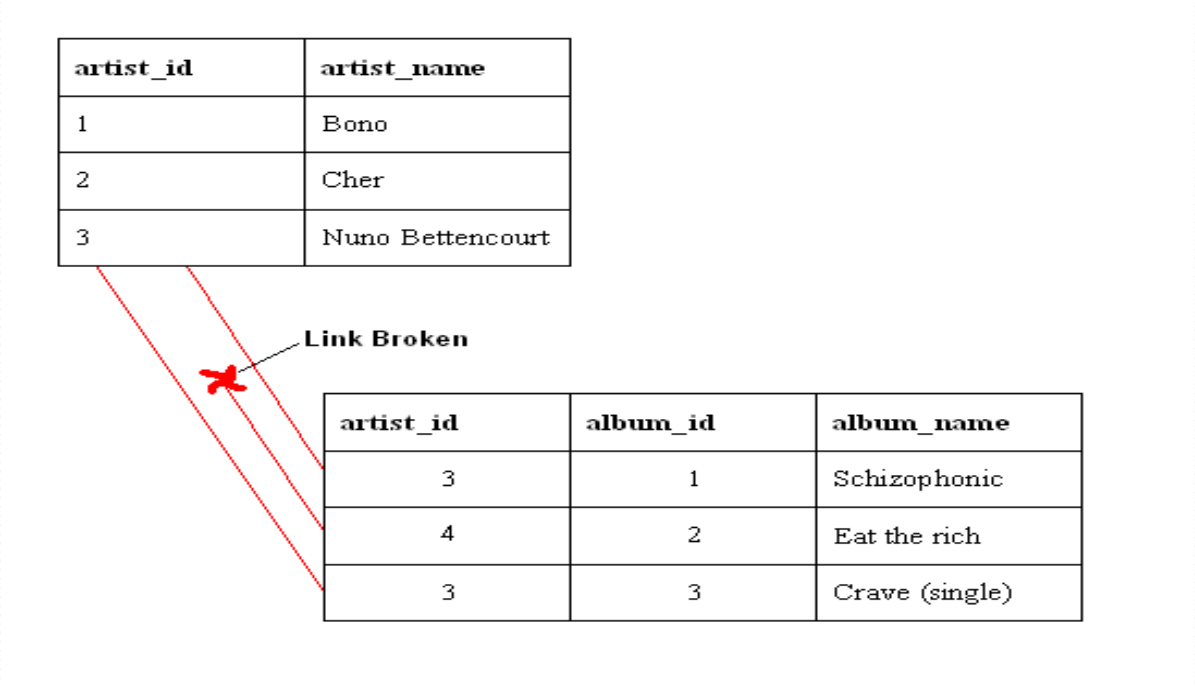
Recursive relationship



No	Name	Title	Manager
1	Ahmed	GM	
2	Ali	Assistant	1
3	Sami	BM	1
4	Hassan	Coordinator	3
5	Sally	Secretary	3
6	Bahaa	Sales Rep.	3
7	Diaa	BM	1
8	Sabrv	Coordinator	7
9	Mona	Secretary	7
10	Gamal	Sales Rep.	7

Referential integrity

- You cannot add a value in a foreign key without a matching value in a primary key.
- it prevents deleting a primary key that there is a foreign key related to it



artist_id	artist_name
1	Bono
2	Cher
3	Nuno Bettencourt

artist_id	album_id	album_name
3	1	Schizophonic
4	2	Eat the rich
3	3	Crave (single)

Database Normalization

Normalization Overview

- **Definition**
  - **the process** of organizing data to minimize redundancy
  - **the process** of decomposing large, inefficiently structured tables into smaller, more efficiently structured tables without losing any data in the process .
  - **the process** of reducing tables to a set of columns where all the non-key columns depend on the primary key column

First normal form

- rules
  - Table must describe only a single object.
  - A single field must not contain multiple data values.
  - Table must not include repeated fields in the same column.
  - Repeated fields must be removed to a related table.
  - Create separate tables for each group of related data.
  - Identify each row with a unique identifier (primary key).

- **Problem**
  - The **CourseCategory** column contains repeated values for the rows.
  - The **CourseInfo** column contains multiple data values in each field
- **Solution**
  - **CourseCategory** > to separate table (Categories)
    - One-Many relationships (Category to Course)
  - **CourseInfo** > to two columns(start date , branch)
    - You need to create branches Tables .

Second normal form

- rules
  - Meet all the requirements of the first normal form.
  - Data in all non-key columns must fully depend on the value of the primary key column or the composite primary key columns.

Example

Customer	Course	CoursePrice
Ahmed	Access	15.00
Bahaa	Access	15.00
Ahmed	Excel	13.00
Mohsen	Word	10.00
Neveen	Oracle	25.00
Zainab	Photo Shop	22.00
Hany	SQL Server	18.00
Hany	Access	15.00
Ahmed	Outlook	9.00
Zainab	Access	15.00
Ayman	Photo Shop	22.00
Rania	Corel Draw	19.00
Rania	SQL Server	18.00
Rania	Access	15.00
Rania	Excel	13.00

- **Problem**
  - CoursePrice depending only on the course column
- **Solution**
  - CoursePrice column > Course Table

Third normal form

- **Rules**
  - Meet all the requirements of the first normal form.
  - Meet all the requirements of the second normal form.
  - Data in all non-key columns must fully describe the value of the primary key column or the composite primary key columns.

Example

BookID	BookName	QuantityInStock	QuantityRequired
1	Access Core	10	6
2	Access Advanced	12	4
3	Excel Applications	8	6
4	Photoshop Core	12	9
5	Windows XP	25	24

- **Problem**
  - QuantityInStock column and the QuantityRequired
- **Solution**
  - QuantityInStock column and the QuantityRequired column must be removed to the book transactions table that fully describes the stock control for the books rather than describing the book itself as an object

Denormalization

- **Definition**
  - the opposite of normalization
- **Why**
  - to optimize the performance of a database
    - if many relations are joined, it may be too slow then to retrieve information

SQL Queries

Working with Oracle 12C

Introduction to Oracle SQL

SQL Commands

- ***Data Definition Language (DDL)***
  - Create , Alter , Drop , Truncate, Replace, Comment
    - deal with the structure of the database objects (the object itself) like tables, views, procedures and so on.
- ***Data Manipulation Language (DML)***
  - Select, Insert , Delete , Update , Merge
    - deal with the contents of the tables rather than the structure of the tables
- ***Data Control Language (DCL)***
  - Grant, Revoke
    - maintain security of the database objects access and use
- ***Transaction Control***
  - Commit, Rollback, Savepoint

Data Definition Language (DDL)

- **CREATE**
  - adding a new database object to the database
- **ALTER**
  - changing the structure of an existing database object
- **DROP**
  - removing a database object from the database permanently
- **TRUNCATE**
  - removing all rows from a table without logging the individual row deletions
- **COMMENT**
  - Add comments to the data dictionary

Data Manipulation Language (DML)

- **SELECT**
  - retrieving data from the database by specifying which columns and rows to be retrieved
- **INSERT**
  - adding a new row (and not column) into the table
- **UPDATE**
  - modifying the existing data in the tabl
- **DELETE**
  - removing an existing data from the table
- **Merge(Upsert)**
  - performs a series of conditional Update and/or Delete and/or Insert operations

Data Control Language (DCL)

- GRANT
  - giving privileges to the users
- REVOKE
  - removing privileges from the users that are previously granted those permissions

Retrieving Data
Customizing Data

SELECT

- Definition
  - Retrieve data from database
- Syntax

```
SELECT * | column1 [, column2, .....]
FROM table
[WHERE conditions]
[ORDER BY column1 [, column2, .....]
ASC, DESC]
```

WHERE

- **Definition**
  - Specify a condition to limit the result
- **Syntax**

```
SELECT  * | column1 [, column2, .....]
FROM table
[WHERE conditions]
```

Order BY

- **Definition**
  - sort the retrieved rows by a specific column or set of columns
- **Syntax**

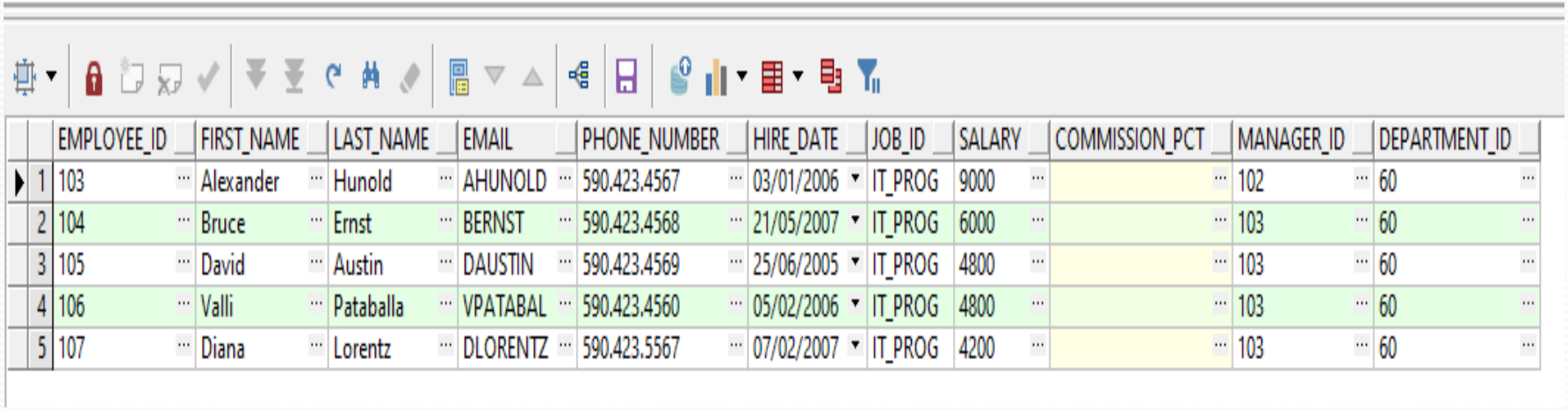
```
SELECT * | column1 [, column2, .....]
FROM table
[WHERE conditions]
[ORDER BY column1 [, column2, .....] ASC, DESC]
```

Example

```
SELECT * FROM EMPLOYEES;

SELECT  EMPLOYEE_ID, FIRST_NAME  FROM EMPLOYEES ;

SELECT EMPLOYEE_ID, FIRST_NAME  FROM EMPLOYEES WHERE SALARY=24000;

SELECT *  FROM EMPLOYEES ORDER BY EMPLOYEE_ID;

SELECT *  FROM EMPLOYEES ORDER BY 1;

SELECT *  FROM EMPLOYEES WHERE SALARY>24000 ORDER BY  MANAGER_ID;
```

Example

```
SELECT *  FROM EMPLOYEES t WHERE JOB_ID='IT_PROG';
```



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	103	Alexander	Hunold	AHUNOLD	590.423.4567	03/01/2006	IT_PROG	9000		102	60
2	104	Bruce	Ernst	BERNST	590.423.4568	21/05/2007	IT_PROG	6000		103	60
3	105	David	Austin	DAUSTIN	590.423.4569	25/06/2005	IT_PROG	4800		103	60
4	106	Valli	Pataballa	VPATABAL	590.423.4560	05/02/2006	IT_PROG	4800		103	60
5	107	Diana	Lorentz	DLORENTZ	590.423.5567	07/02/2007	IT_PROG	4200		103	60

Example

```
SELECT *  FROM EMPLOYEES T ORDER BY SALARY DESC;
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	17/06/2003	AD_PRES	24000
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/2005	AD_VP	17000
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13/01/2001	AD_VP	17000
4	145	John	Russell	JRUSSEL	011.44.1344.429268	01/10/2004	SA_MAN	14000
5	146	Karen	Partners	KPARTNER	011.44.1344.467268	05/01/2005	SA_MAN	13500
6	201	Michael	Hartstein	MHARTSTE	515.123.5555	17/02/2004	MK_MAN	13000
7	108	Nancy	Greenberg	NGREENBE	515.124.4569	17/08/2002	FI_MGR	12008
8	205	Shelley	Higgins	SHIGGINS	515.123.8080	07/06/2002	AC_MGR	12008
9	147	Alberto	Errazuriz	AERRAZUR	011.44.1344.429278	10/03/2005	SA_MAN	12000
10	168	Lisa	Ozer	LOZER	011.44.1343.929268	11/03/2005	SA_REP	11500
11	114	Den	Raphaely	DRAPHEAL	515.127.4561	07/12/2002	PU_MAN	11000
12	148	Gerald	Cambrault	GCAMBRAU	011.44.1344.619268	15/10/2007	SA_MAN	11000
13	174	Ellen	Abel	EABEL	011.44.1644.429267	11/05/2004	SA_REP	11000
14	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29/01/2008	SA_MAN	10500
15	162	Clara	Vishney	CVISHNEY	011.44.1346.129268	11/11/2005	SA_REP	10500
16	156	Janette	King	JKING	011.44.1345.429268	30/01/2004	SA_REP	10000

SQL AND & OR Operators

- **AND**
  - The AND operator displays a record if both the first condition AND the second condition are true.
- **OR**
  - The OR operator displays a record if either the first condition OR the second condition is true.

```
SELECT *
  FROM EMPLOYEES
 WHERE LAST_NAME = 'King'
   AND JOB_ID = 'IT_PROG';

SELECT *
  FROM EMPLOYEES
 where JOB_ID = 'SA_REP'
    OR JOB_ID = 'IT_PROG';
```

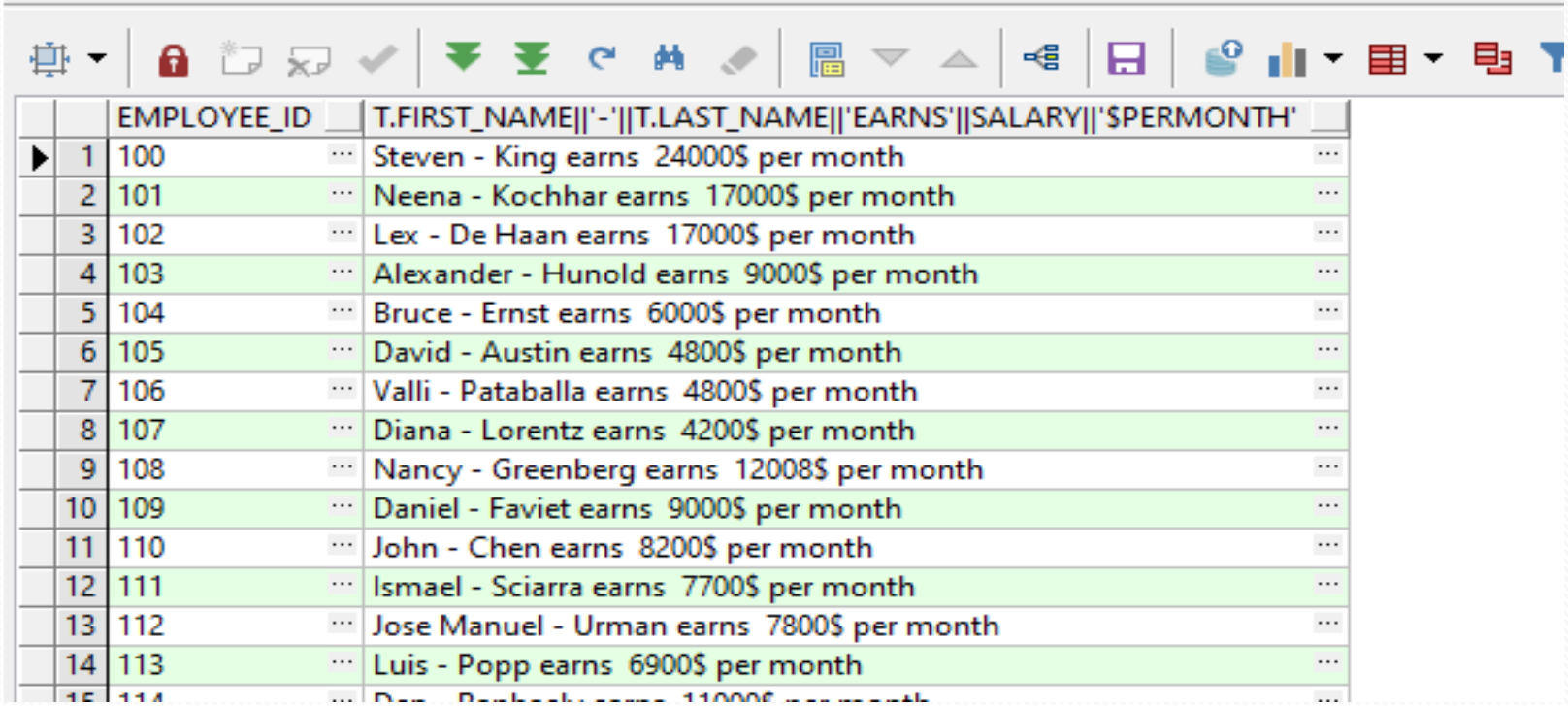
NULL

- In the database world, NULL is special. It is a marker for missing information or the information is not applicable

```
SELECT * FROM EMPLOYEES WHERE MANAGER_ID IS NULL;

SELECT * FROM EMPLOYEES WHERE MANAGER_ID IS NOT NULL;

SELECT EMPLOYEE_ID, COMMISSION_PCT, NVL(COMMISSION_PCT, 0)
  FROM EMPLOYEES t;
```

Concatenate

```
SELECT EMPLOYEE_ID,
       T.FIRST_NAME || ' - ' || T.LAST_NAME || ' earns  ' || SALARY ||
       '$ per month'
  FROM EMPLOYEES T;
```



	EMPLOYEE_ID	T.FIRST_NAME||'-'||T.LAST_NAME||'EARNS'||SALARY||'$PERMONTH'
1	100	Steven - King earns 24000$ per month
2	101	Neena - Kochhar earns 17000$ per month
3	102	Lex - De Haan earns 17000$ per month
4	103	Alexander - Hunold earns 9000$ per month
5	104	Bruce - Ernst earns 6000$ per month
6	105	David - Austin earns 4800$ per month
7	106	Valli - Pataballa earns 4800$ per month
8	107	Diana - Lorentz earns 4200$ per month
9	108	Nancy - Greenberg earns 12008$ per month
10	109	Daniel - Faviet earns 9000$ per month
11	110	John - Chen earns 8200$ per month
12	111	Ismael - Sciarra earns 7700$ per month
13	112	Jose Manuel - Urman earns 7800$ per month
14	113	Luis - Popp earns 6900$ per month

Aliases (as)

```
SELECT EMPLOYEE_ID, FIRST_NAME || ' - ' || LAST_NAME    as FULLNAME
  FROM EMPLOYEES ;
```

	EMPLOYEE_ID	FULLNAME
1	100	Steven - King
2	101	Neena - Kochhar
3	102	Lex - De Haan
4	103	Alexander - Hunold
5	104	Bruce - Ernst
6	105	David - Austin
7	106	Valli - Pataballa
8	107	Diana - Lorentz
9	108	Nancy - Greenberg
10	109	Daniel - Faviet
11	110	John - Chen
12	111	Ismael - Sciarra
13	112	Jose Manuel - Urman
14	113	Luis - Popp

Distinct

- eliminates duplicate row values from the results

```
SELECT   JOB_ID    FROM EMPLOYEES ;
```

1

```
SELECT DISTINCT    JOB_ID     FROM EMPLOYEES ;
```

2

1

	JOB_ID
1	AC_ACCOUNT
2	AC_MGR
3	AD_ASST
4	AD_PRES
5	AD_VP
6	AD_VP
7	FI_ACCOUNT
8	FI_ACCOUNT
9	FI_ACCOUNT
10	FI_ACCOUNT
11	FI_ACCOUNT
12	FI_MGR
13	HR_REP
14	IT_PROG
15	IT_PROG
16	IT_PROG
17	IT_PROG
18	IT_PROG
19	MK_MAN
20	MK_REP
21	PR_REP
22	PU_CLERK

2

	JOB_ID
1	AC_ACCOUNT
2	AC_MGR
3	AD_ASST
4	AD_PRES
5	AD_VP
6	FI_ACCOUNT
7	FI_MGR
8	HR_REP
9	IT_PROG
10	MK_MAN
11	MK_REP
12	PR_REP
13	PU_CLERK
14	PU_MAN
15	SA_MAN
16	SA_REP
17	SH_CLERK
18	ST_CLERK
19	ST_MAN

Grouping Data

Grouping Functions

Function	Description
SUM	Returns the summation of numeric values
AVG	Returns the average of numeric values
MIN	Returns the lowest value from a set of values
MAX	Return the highest value from a set of values
COUNT	Return the total number of a set of values

Example 1

```
SELECT SUM(SALARY) FROM EMPLOYEES;
SELECT MIN(SALARY) FROM EMPLOYEES;
SELECT MAX(SALARY) FROM EMPLOYEES;
SELECT AVG(SALARY) FROM EMPLOYEES;
SELECT COUNT(*) FROM EMPLOYEES;
```

Select employees | Select employees | Select employees | Select employees | Select employees

	SUM(SALARY)
1	691416

	MIN(SALARY)
1	2100

	MAX(SALARY)
1	24000

	AVG(SALARY)
1	6461.831775700934579439252336448598130841

	COUNT(*)
1	107

Example 2

```
SELECT COUNT(*) FROM EMPLOYEES;
SELECT COUNT(MANAGER_ID) FROM EMPLOYEES;
SELECT COUNT(EMPLOYEE_ID) FROM EMPLOYEES;
```

Select employees | Select employees | Select employees

	COUNT(*)
1	107

	COUNT(MANAGER_ID)
1	106

	COUNT(EMPLOYEE_ID)
1	107

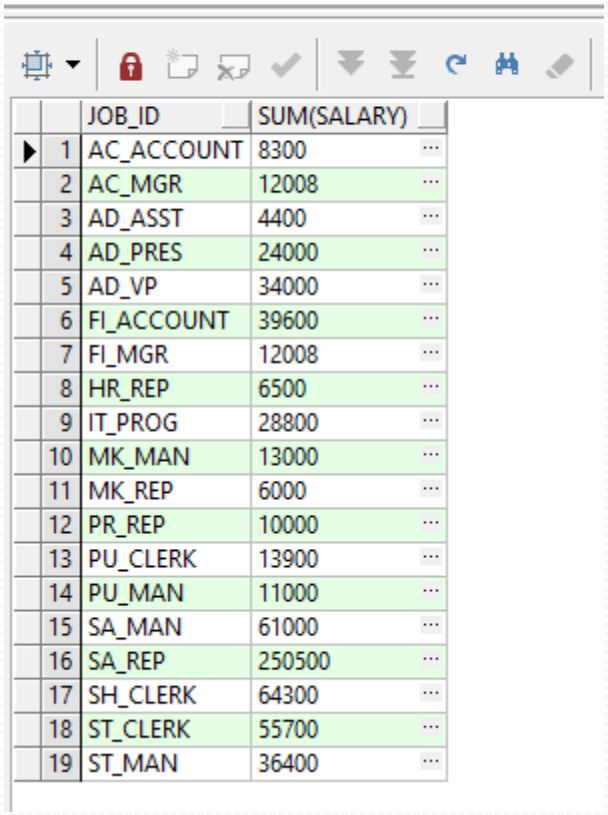
Group By

- **Definition**
  - used to divide the rows in a table into smaller groups
- **Syntax**

```
SELECT  column1, group_function (column2),
.....
FROM table
[WHERE conditions]
[GROUP BY column1, ........]
[ORDER BY column1 [, column2, ........] ASC, DESC]
```

Example : Group By

```
SELECT JOB_ID, SUM(SALARY)
FROM EMPLOYEES
GROUP BY JOB_ID
ORDER BY JOB_ID;
```



	JOB_ID	SUM(SALARY)
1	AC_ACCOUNT	8300
2	AC_MGR	12008
3	AD_ASST	4400
4	AD_PRES	24000
5	AD_VP	34000
6	FI_ACCOUNT	39600
7	FI_MGR	12008
8	HR_REP	6500
9	IT_PROG	28800
10	MK_MAN	13000
11	MK_REP	6000
12	PR_REP	10000
13	PU_CLERK	13900
14	PU_MAN	11000
15	SA_MAN	61000
16	SA_REP	250500
17	SH_CLERK	64300
18	ST_CLERK	55700
19	ST_MAN	36400

Having

- **Definition**
  - used to restrict the group results
- **Syntax**

```
SELECT * | column1, group_function (column2),
.....
FROM table
[WHERE conditions]
[GROUP BY column1, ........]
HAVING [conditions]
[ORDER BY column1 [, column2, ........] ASC, DESC]
```

Example : Having

```
SELECT JOB_ID, SUM(SALARY)
FROM EMPLOYEES
GROUP BY JOB_ID
HAVING SUM(SALARY)>9000
ORDER BY JOB_ID;
```

	JOB_ID	SUM(SALARY)
1	AC_MGR	12008
2	AD_PRES	24000
3	AD_VP	34000
4	FI_ACCOUNT	39600
5	FI_MGR	12008
6	IT_PROG	28800
7	MK_MAN	13000
8	PR_REP	10000
9	PU_CLERK	13900
10	PU_MAN	11000
11	SA_MAN	61000
12	SA_REP	250500
13	SH_CLERK	64300
14	ST_CLERK	55700
15	ST_MAN	36400

SQL Operators

Using SQL Operators

- Arithmetic operators
- Comparison operators
- Logical operators
- Set Operators
- Other operators

Arithmetic operators

- addition (+)
- subtraction (-)
- multiplication (*)
- division (/).

Comparison operators

- compare two or more values

Operator	Description
=	Equal
<	Lease than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
<>	Not Equal

Example

SQL | Output | Statistics

```
SELECT * FROM EMPLOYEES WHERE SALARY > 15000;
SELECT * FROM EMPLOYEES WHERE SALARY < 2400;
SELECT * FROM EMPLOYEES WHERE MANAGER_ID = 100;
SELECT * FROM EMPLOYEES WHERE MANAGER_ID <> 100;
```

Select employees | Select employees | Select employees | Select employees

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	17/06/2003	AD_PRES	24000			90
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/2005	AD_VP	17000		100	90
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13/01/2001	AD_VP	17000		100	90

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	128	Steven	Markle	SMARKLE	650.124.1434	08/03/2008	ST_CLERK	2200		120	50
2	132	TJ	Olson	TJOLSON	650.124.8234	10/04/2007	ST_CLERK	2100		121	50
3	136	Hazel	Philtanker	HPHILTAN	650.127.1634	06/02/2008	ST_CLERK	2200		122	50

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/2005	AD_VP	17000		100	90
2	102	Lex	De Haan	LDEHAAN	515.123.4569	13/01/2001	AD_VP	17000		100	90
3	114	Den	Raphaely	DRAPHEAL	515.127.4561	07/12/2002	PU_MAN	11000		100	30
4	120	Matthew	Weiss	MWEISS	650.123.1234	18/07/2004	ST_MAN	8000		100	50
5	121	Adam	Fripp	AFRIPP	650.123.2234	10/04/2005	ST_MAN	8200		100	50
6	122	Payam	Kaufling	PKAUFLIN	650.123.3234	01/05/2003	ST_MAN	7900		100	50
7	123	Shanta	Vollman	SVOLLMAN	650.123.4234	10/10/2005	ST_MAN	6500		100	50

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	103	Alexander	Hunold	AHUNOLD	590.423.4567	03/01/2006	IT_PROG	9000		102	60
2	104	Bruce	Ernst	BERNST	590.423.4568	21/05/2007	IT_PROG	6000		103	60
3	105	David	Austin	DAUSTIN	590.423.4569	25/06/2005	IT_PROG	4800		103	60
4	106	Valli	Pataballa	VPATABAL	590.423.4560	05/02/2006	IT_PROG	4800		103	60
5	107	Diana	Lorentz	DLORENTZ	590.423.5567	07/02/2007	IT_PROG	4200		103	60
6	108	Nancy	Greenberg	NGREENBE	515.124.4569	17/08/2002	FI_MGR	12008		101	100

Logical operators

- Used to get a logical value (True or False)

Operator	Result
AND	Return True only if both or all expressions are True
OR	Return True if at least one expression is True
NOT	Return True if expression is False and vice versa

Example

SQL | Output | Statistics

```
SELECT * FROM EMPLOYEES WHERE SALARY > 15000 and MANAGER_ID = 100;

SELECT * FROM EMPLOYEES WHERE FIRST_NAME='Neena' OR JOB_ID='IT_PROG';

SELECT * FROM EMPLOYEES WHERE SALARY <= 15000 and SALARY >= 13000;
```

Select employees | Select employees | Select employees

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/2005	AD_VP	17000		100	90
2	102	Lex	De Haan	LDEHAAN	515.123.4569	13/01/2001	AD_VP	17000		100	90

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/2005	AD_VP	17000		100	90
2	103	Alexander	Hunold	AHUNOLD	590.423.4567	03/01/2006	IT_PROG	9000		102	60
3	104	Bruce	Ernst	BERNST	590.423.4568	21/05/2007	IT_PROG	6000		103	60
4	105	David	Austin	DAUSTIN	590.423.4569	25/06/2005	IT_PROG	4800		103	60
5	106	Valli	Pataballa	VPATABAL	590.423.4560	05/02/2006	IT_PROG	4800		103	60
6	107	Diana	Lorentz	DLORENTZ	590.423.5567	07/02/2007	IT_PROG	4200		103	60

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	145	John	Russell	JRUSSEL	011.44.1344.429268	01/10/2004	SA_MAN	14000	.4	100	80
2	146	Karen	Partners	KPARTNER	011.44.1344.467268	05/01/2005	SA_MAN	13500	.3	100	80
3	201	Michael	Hartstein	MHARTSTE	515.123.5555	17/02/2004	MK_MAN	13000		100	20

Set Operators

- **Definition**
  - combine the results of two or more queries into one result
- **Keywords**
  - UNION/ UNION ALL
  - INTERSECT
  - MINUS

UNION & Intersect & Minus

EmpLacation
Egypt
Canada
Egypt

VacLocation
Egypt
Canada
France

Union
Egypt
Canada
France

Intersect
Egypt
Canada

Except
France

UNION/UNION ALL Operator

- Create a Single Result Set from Multiple Queries
- Each Query Must Have:
  - Similar data types
  - Same number of columns
  - Same column order in select list
  - The UNION operator returns only distinct rows that appear in either result

INTERSECT Operator

- **Definition**
  - returns only the records that have the same values in the selected columns in both tables.
- **Syntax**

```
USE northwind
SELECT  (firstname + ' ' + lastname) AS name
       ,city, postalcode
 FROM employees
INTERSECT
SELECT companyname, city, postalcode
 FROM customers
GO
```

EXCEPT Operator

- **Definition**
  - return rows returned by the first query that are not present in the second query
- **Syntax**

```
USE northwind
SELECT  (firstname + ' ' + lastname) AS name
       ,city, postalcode
 FROM employees
EXCEPT
SELECT companyname, city, postalcode
 FROM customers
GO
```

Other operators

Operator	Description
IS	Representing null values
LIKE	Searching by using wild cards (% and _)
IN	Searching in a set of values
BETWEEN	Searching in a range of values

SQLQuery4.sql ...LOZ\Lolo (54)) | SQLQuery1.sql ...OZ\Lolo (53))*

```
select  OrderDate, ShipName, Freight, ShipRegion
from Orders
where ShipRegion is Null
```

Results | Messages

	OrderDate	ShipName	Freight	ShipRegion
1	1996-07-04 00:00:00.000	Vins et alcools Chevalier	32.38	NULL
2	1996-07-05 00:00:00.000	Toms Spezialitäten	11.61	NULL
3	1996-07-08 00:00:00.000	Victuailles en stock	41.34	NULL
4	1996-07-09 00:00:00.000	Suprêmes délices	51.30	NULL
5	1996-07-11 00:00:00.000	Chop-suey Chinese	22.98	NULL
6	1996-07-12 00:00:00.000	Richter Supermarkt	148.33	NULL
7	1996-07-17 00:00:00.000	Ernst Handel	140.51	NULL
8	1996-07-18 00:00:00.000	Centro comercial Moctezuma	3.25	NULL
9	1996-07-19 00:00:00.000	Ottilies Käseladen	55.09	NULL
10	1996-07-23 00:00:00.000	Ernst Handel	146.06	NULL
11	1996-07-24 00:00:00.000	Folk och fä HB	3.67	NULL
12	1996-07-25 00:00:00.000	Blondel père et fils	55.28	NULL
13	1996-07-26 00:00:00.000	Wartian Herkku	25.73	NULL
14	1996-07-29 00:00:00.000	Frankenversand	208.58	NULL

SQLQuery4.sql ...LOZ\Lolo (54)) | SQLQuery1.sql ...OZ\Lolo (53))*

```
select  OrderDate, ShipName, Freight, ShipRegion
from Orders
where Freight between 500 and 1000
```

Results | Messages

	OrderDate	ShipName	Freight	ShipRegion
1	1996-12-04 00:00:00.000	Queen Cozinha	890.78	SP
2	1997-03-19 00:00:00.000	Rattlesnake Canyon Grocery	708.95	NM
3	1997-04-22 00:00:00.000	Ernst Handel	789.95	NULL
4	1997-07-28 00:00:00.000	Save-a-lot Markets	544.08	ID
5	1997-10-03 00:00:00.000	QUICK-Stop	810.05	NULL
6	1998-01-06 00:00:00.000	Great Lakes Food Market	719.78	OR
7	1998-02-19 00:00:00.000	Hungry Owl All-Night Grocers	603.54	Co. Cork
8	1998-02-26 00:00:00.000	Hungry Owl All-Night Grocers	580.91	Co. Cork
9	1998-03-27 00:00:00.000	Save-a-lot Markets	657.54	ID
10	1998-04-13 00:00:00.000	Ernst Handel	754.26	NULL
11	1998-04-17 00:00:00.000	Save-a-lot Markets	830.75	ID
12	1998-04-17 00:00:00.000	White Clover Markets	606.19	WA

SQLQuery4.sql ...LOZ\Lolo (54)) | SQLQuery1.sql ...OZ\Lolo (53))*

```
select  EmployeeID, LastName, Title
from Employees
where LastName like '_u%'
```

Results | Messages

	EmployeeID	LastName	Title
1	2	Fuller	Vice President, Sales
2	5	Buchanan	Sales Manager
3	6	Suyama	Sales Representative

SQLQuery4.sql ...LOZ\Lolo (54)) | **SQLQuery1.sql ...OZ\Lolo (53))***

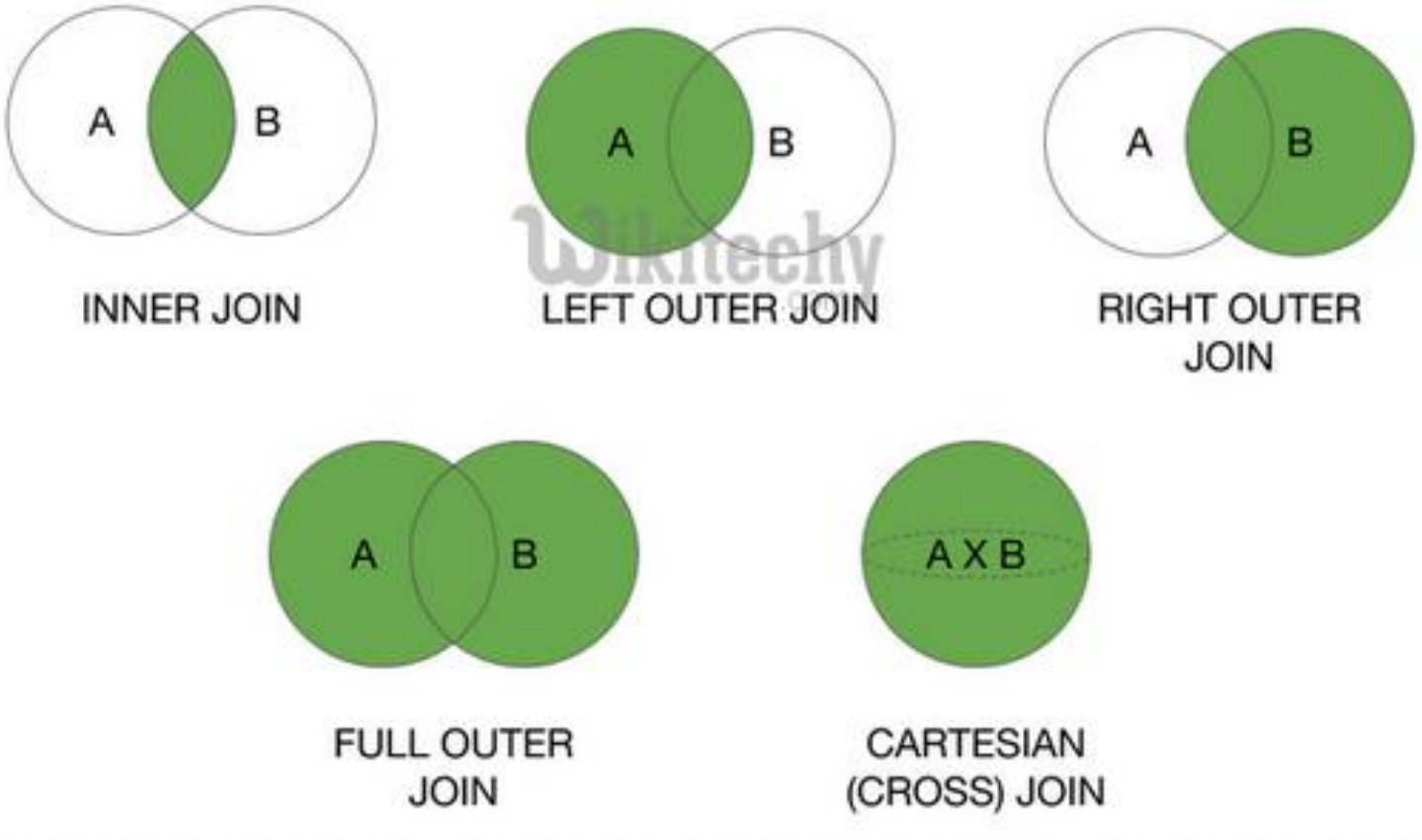
```
select   EmployeeID , LastName, City
from Employees


select   EmployeeID , LastName, City
from Employees
where City in ('London', 'Tacoma')
```

Results | Messages

	EmployeeID	LastName	City
1	1	Davolio	Seattle
2	2	Fuller	Tacoma
3	3	Leverling	Kirkland
4	4	Peacock	Redmond
5	5	Buchanan	London
6	6	Suyama	London
7	7	King	London
8	8	Callahan	Seattle
9	9	Dodsworth	London

	EmployeeID	LastName	City
1	2	Fuller	Tacoma
2	5	Buchanan	London
3	6	Suyama	London
4	7	King	London
5	9	Dodsworth	London

Joining Data


A
B
INNER JOIN
A
B
LEFT OUTER JOIN
A
B
RIGHT OUTER JOIN
A
B
FULL OUTER JOIN
A X B
CARTESIAN (CROSS) JOIN

Using Aliases for Table Names

- Example 1 (without an alias name)

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

- Example 2 (with an alias name)

```
SELECT c.CustomerName, Or.OrderID
FROM Customers as c
INNER JOIN Orders as Or
ON c.CustomerID=Or.CustomerID
ORDER BY c.CustomerName;
```

This slide from 2071b from Microsoft

Combining Data from Multiple Tables

- Introduction to Joins
- Using Inner Joins
- Using Outer Joins
- Using Cross Joins
- Joining More Than Two Tables
- Joining a Table to Itself

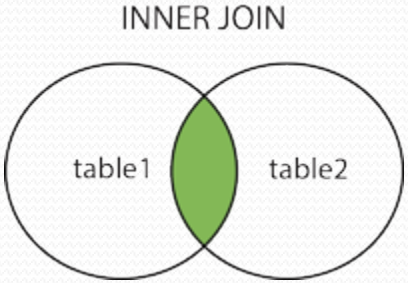
Introduction to Joins

- Selects Specific Columns from Multiple Tables
  - JOIN keyword specifies that tables are joined and how to join them
  - ON keyword specifies join condition
- Queries Two or More Tables to Produce a Result Set
  - Use primary and foreign keys as join conditions
  - Use columns common to specified tables to join tables

Inner Join

- **Definition**
  - return rows only when there is at least one row from both tables that matches the join condition
- **Syntax**

```
SELECT <select list>
FROM table1 inner join table2
ON (table1.column1 = table2.column2)
```



Using Inner Joins

```
USE joindb
SELECT buyer_name, sales.buyer_id, qty
FROM buyers  INNER JOIN sales
ON buyers.buyer_id = sales.buyer_id
GO
```

Example 1

buyers

buyer_name	*buyer_id*
Adam Barr	1
Sean Chai	2
Eva Corets	3
Erin O'Melia	4

sales

buyer_id	*prod_id*	*qty*
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

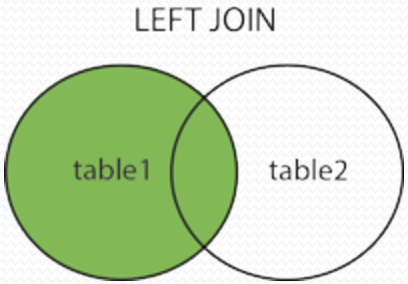
buyer_name	*buyer_id*	*qty*
Adam Barr	1	15
Adam Barr	1	5
Erin O'Melia	4	37
Eva Corets	3	11
Erin O'Melia	4	1003

Left Outer Join

- **Definition**
  - includes all rows in the left table in the results whether or not there are matching values on the common column in the right table. in addition to the matching values in Both table.
  - **In other words**: All rows from the Left table are returned
- **Syntax**

```
SELECT <select list>
FROM table1 left outer join table2
ON (table1.column1 = table2.column2)
```

- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  LEFT JOIN Orders
  ON Customers.CustomerID=Orders.CustomerID
  ORDER BY Customers.CustomerName;



Using Outer Joins

```
USE joindb
SELECT buyer_name, sales.buyer_id, qty
 FROM buyers  LEFT OUTER JOIN sales
  ON buyers.buyer_id = sales.buyer_id
GO
```

Example 1

buyers

buyer_name	*buyer_id*
Adam Barr	1
Sean Chai	2
Eva Corets	3
Erin O'Melia	4

sales

buyer_id	*prod_id*	*qty*
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer_name	*buyer_id*	*qty*
Adam Barr	1	15
Adam Barr	1	5
Erin O'Melia	4	37
Eva Corets	3	11
Erin O'Melia	4	1003
Sean Chai	NULL	NULL

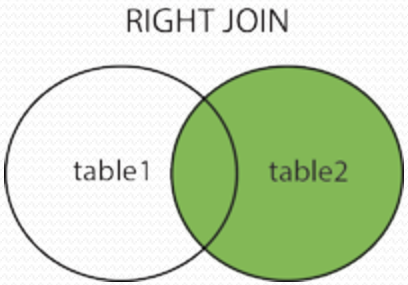
Right Outer join

- **Definition**
  - the reverse of LEFT OUTER joins
  - All rows from the right table are returned
    - Null values are returned for the left table any time a right table row has no matching row in the left table.
- **Syntax**

```
SELECT <select list>
FROM table1 right outer join table2
ON (table1.column1 =
table2.column2)
```

- SELECT Orders.OrderID, Employees.FirstName
  FROM Orders
  RIGHT JOIN Employees
  ON Orders.EmployeeID=Employees.EmployeeID
  ORDER BY Orders.OrderID;

In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



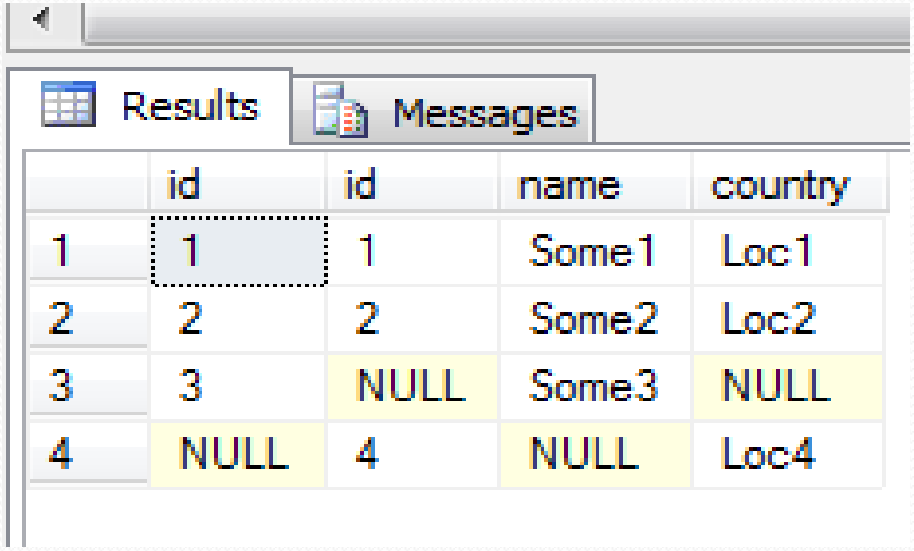
Full Join

- returns all the rows from the left table , and all the rows from the right table
- Also it returns rows in the left table that do not have matches in the right table, or if there are rows in right table that do not have matches in the left table.

ID	Name
1	Some1
2	Some2
3	Some3

ID	Country
1	Loc1
2	Loc2
4	Loc4

```
SELECT a.id ,b.id, a.name , b.country
FROM LeftTable as a
Full Join RightTable as b
On a.id = b.id
```



	id	id	name	country
1	1	1	Some1	Loc1
2	2	2	Some2	Loc2
3	3	NULL	Some3	NULL
4	NULL	4	NULL	Loc4

Full Join

Self- Join

- **Definition**
  - Join table with it self
- **Syntax**

```
SELECT  a.column , b.column
FROM Table1 as a
JOIN Table1 as b
ON a.column = b.column
```

Problem

- **I want Every Employee Name with his Manager**

No	Name	Title	Manager
1	Ahmed	GM	
2	Ali	Assistant	1
3	Sami	BM	1
4	Hassan	Coordinator	3
5	Sally	Secretary	3
6	Bahaa	Sales Rep.	3
7	Diaa	BM	1
8	Sabry	Coordinator	7
9	Mona	Secretary	7
10	Gamal	Sales Rep.	7

Solution

SQLQuery5.sql - A...d-PC\ahmed (53))*

```
use Testdb
SELECT   EmployeesA.Name as EmployeeName
       , ManagersB.Name as ManagerName
from EmployeesMangers as ManagersB
Join EmployeesMangers as EmployeesA
ON ManagersB.ID = EmployeesA.Manager
```

Results | Messages

	EmployeeName	ManagerName
1	Ahmed	Ahmed
2	ALi	Ahmed
3	Sami	Ahmed
4	Hassan	Sami
5	sally	Sami
6	Bahaa	Sami
7	Diaa	Ahmed
8	sabry	Diaa
9	Mona	Diaa
10	Gamal	Diaa

Cross Join

- return all rows from both tables
- Each row from the left table is combined with all rows from Second Table
- the number of rows in the left table multiplied by the number of rows in the right table.
  - **Table1 >> 2 rows,**
  - **Table2 >> 5 rows,**
  - **Result >>10 rows**
- **Example** :
  - Possible ways

Name
Ali
Islam

Country
Egypt
Cairo

Name	Country
Ali	Egypt
Ali	Cairo
Islam	Egypt
Islam	Cairo

Cross Join

Using Cross Joins

```
USE joindb
SELECT buyer_name, qty
 FROM buyers
 CROSS JOIN sales
GO
```

Example 1

buyers

buyer_id	buyer_name
1	Adam Barr
2	Sean Chai
3	Eva Corets
4	Erin O'Melia

sales

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer_name	qty
Adam Barr	15
Adam Barr	5
Adam Barr	37
Adam Barr	11
Adam Barr	1003
Sean Chai	15
Sean Chai	5
Sean Chai	37
Sean Chai	11
Sean Chai	1003
Eva Corets	15
...	...

Example

- All of the possible ways that suppliers can ship their products

SQLQuery6.sql - ...d-PC\ahmed (52))*

```
USE Northwind
SELECT Suppliers.CompanyName ,Shippers.CompanyName
 FROM  Suppliers
 CRoss Join  Shippers
```

Results | Messages

	CompanyName	CompanyName
1	Exotic Liquids	Speedy Express
2	New Orleans Cajun Delights	Speedy Express
3	Grandma Kelly's Homestead	Speedy Express
4	Tokyo Traders	Speedy Express
5	Cooperativa de Quesos 'Las Cabras'	Speedy Express
6	Mayumi's	Speedy Express
7	Pavlova, Ltd.	Speedy Express
8	Specialty Biscuits, Ltd.	Speedy Express
9	PB Kn?ckebr?d AB	Speedy Express
10	Refrescos Americanas LTDA	Speedy Express
11	Heli Sü?waren GmbH & Co. KG	Speedy Express
12	Plutzer Lebensmittelgro?m?rkte AG	Speedy Express
13	Nord-Ost-Fisch Handelsgesellscha...	Speedy Express
14	Formaggi Fortini s.r.l.	Speedy Express
15	Norske Meierier	Speedy Express

CASE is used to provide if-then-else type of logic to SQL. There are two formats: The first is a **Simple CASE** expression, where we compare an expression to static values. The second is a

- **Searched CASE** expression, where we compare an expression to one or more logical conditions.
- **Simple CASE Expression**
- The syntax for a **simple CASE** expression is:
- **SELECT CASE** ("**column_name**")
  WHEN "**value1**" **THEN** "**result1**"
  WHEN "**value2**" **THEN** "**result2**"
  ...
  [**ELSE** "**resultN**"]
  END
  FROM "**table_name**";

Subquery

- In Oracle, a subquery is a query within a query. You can create subqueries within your SQL statements. These subqueries can reside in the **WHERE** clause, the **FROM** clause, or the **SELECT** clause

- When a query is included inside another query, the Outer query is known as **Main Query**, and Inner query is known as **Subquery**

By: geeksforgeeks

Subqueries...

- **WHERE clause**

```
SELECT *
  FROM HR.EMPLOYEES EMP
 WHERE EMP.DEPARTMENT_ID IN
       (SELECT DEP.DEPARTMENT_ID
          FROM HR.DEPARTMENTS DEP
         WHERE DEP.DEPARTMENT_NAME = 'Shipping');
```

- **FROM clause**

```
SELECT DEP.DEPARTMENT_NAME, SUBQUERY1.TOTAL_SALARY
  FROM HR.DEPARTMENTS DEP
  JOIN (SELECT EMP.DEPARTMENT_ID, SUM(EMP.SALARY) AS TOTAL_SALARY
          FROM HR.EMPLOYEES EMP
         GROUP BY EMP.DEPARTMENT_ID) SUBQUERY1
    ON DEP.DEPARTMENT_ID = SUBQUERY1.DEPARTMENT_ID;
```

- **SELECT clause (Scalar Subqueries)**

```
SELECT DEP.DEPARTMENT_NAME,
       (SELECT COUNT(EMP.EMPLOYEE_ID) AS TOTAL_EMPLOYEES
          FROM HR.EMPLOYEES EMP
         WHERE EMP.DEPARTMENT_ID = DEP.DEPARTMENT_ID) SUBQUERY2
  FROM HR.DEPARTMENTS DEP
```

By: geeksforgeeks

Subquery

- **Nested Query**
  - In Nested Query, Inner query runs first, and only once. Outer query is executed with result from Inner query. Hence, Inner query is used in execution of Outer query

Example:

```
SELECT *
  FROM HR.EMPLOYEES EMP
 WHERE EMP.EMPLOYEE_ID IN
        (SELECT JOB.EMPLOYEE_ID FROM HR.JOB_HISTORY JOB);
```

By: geeksforgeeks

Subquery

- **Correlated Query**
  - In Correlated Query, Outer query executes first and for every Outer query row Inner query is executed. Hence, Inner query uses values from Outer query

Example:

```
SELECT *
    FROM HR.EMPLOYEES EMP
   WHERE EXISTS (SELECT JOB.EMPLOYEE_ID
                 FROM HR.JOB_HISTORY JOB
                WHERE JOB.EMPLOYEE_ID = EMP.EMPLOYEE_ID);
```

By: geeksforgeeks

Join Operation

- **Join Operation**
  - Join operation is a binary operation used to combine data or rows from two or more tables based on a common field between them. INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN are different types of Joins

Example:

```
SELECT EMP.*
  FROM HR.EMPLOYEES EMP
  JOIN HR.JOB_HISTORY JOB
    ON JOB.EMPLOYEE_ID = EMP.EMPLOYEE_ID;
```

By: geeksforgeeks

Subquery and Join Operation

- **Nested Query**

```
SELECT *
  FROM HR.EMPLOYEES EMP
 WHERE EMP.EMPLOYEE_ID IN
       (SELECT JOB.EMPLOYEE_ID FROM HR.JOB_HISTORY JOB);
```

- **Correlated Query**

```
SELECT *
   FROM HR.EMPLOYEES EMP
  WHERE EXISTS (SELECT JOB.EMPLOYEE_ID
           FROM HR.JOB_HISTORY JOB
          WHERE JOB.EMPLOYEE_ID = EMP.EMPLOYEE_ID);
```

- **Join Operation**

```
SELECT EMP.*
  FROM HR.EMPLOYEES EMP
  JOIN HR.JOB_HISTORY JOB
    ON JOB.EMPLOYEE_ID = EMP.EMPLOYEE_ID;
```

By: geeksforgeeks

Subquery and Join Operation

Parameters	Nested Query	Correlated Query	Join Operation
Definition	In Nested query, a query is written inside another query and the result of inner query is used in execution of outer query.	In Correlated query, a query is nested inside another query and inner query uses values from outer query.	Join operation is used to combine data or rows from two or more tables based on a common field between them.INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN are different types of Joins.
Approach	Bottom up approach i.e. Inner query runs first, and only once. Outer query is executed with result from Inner query.	Top to Down Approach i.e. Outer query executes first and for every Outer query row Inner query is executed.	It is basically cross product satisfying a condition.
Dependency	Inner query execution is not dependent on Outer query.	Inner query is dependent on Outer query.	There is no Inner Query or Outer Query. Hence, no dependency is there.
Performance	Performs better than Correlated Query but is slower than Join Operation.	Performs slower than both Nested Query and Join operations as for every outer query inner query is executed.	By using joins we maximize the calculation burden on the database but joins are better optimized by the server so the retrieval time of the query using joins will almost always be faster than that of a subquery.

By: geeksforgeeks