Mu Sigma

MuCell – SQL Intermediate

Do The Math

Chicago, IL
Bangalore, India
www.mu-sigma.com

February, 2014

Agenda for the training

- Data Modification Statements
- Functions
- Window Functions (RANK, ROW NUMBER, NTILE, DENSE_RANK)
- Stored Procedures
- Views
- Indexes

INSERT Statement adds rows to a table

- The **INSERT** Statement adds one or more rows to a table
- It has two formats:

```
1. INSERT INTO table_name[(column-list)] VALUES (value-list)
```

```
2. INSERT INTO table_name [(column-list)] (query-specification)
```

- Example:

```
1) INSERT INTO dbo.intr_emp (firstname, lastname)
   VALUES (‘Sutanu', ‘Kandar')

2) INSERT INTO dbo.intr_emp
   SELECT firstname, lastname
   FROM    dbo.DimCustomer
   WHERE Customerkey=4
```

Source Table

S.NO	First Name	Last Name
1	Shashank	Sharma
2	Arun	Kumar
3	Shyam	Shankar

Output Table

S.NO	First Name	Last Name
1	Shashank	Sharma
2	Arun	Kumar
3	Shyam	Shankar
4	Sutanu	Kandar

UPDATE Statement modify columns in table rows

- The **UPDATE** statement modifies columns in selected table rows

- It has following general format:

```
UPDATE table_name SET set-list [WHERE clause]
```

- Example:

```
UPDATE dbo.intr_emp
SET lastname='Raman'
WHERE firstname='Arun'
```

Source Table

S.NO	First Name	Last Name
1	Shashank	Sharma
2	Arun	Kumar
3	Shyam	Shankar

Output Table

S.NO	First Name	Last Name
1	Shashank	Sharma
2	Arun	**Raman**
3	Shyam	Shankar

DELETE Statement remove rows from tables

- The **DELETE** Statement removes selected rows from a table
- It has following general format:

```
DELETE FROM table_name  [WHERE clause]
```

- Example:

```
DELETE FROM dbo.intr_emp
WHERE firstname='Sutanu'
```

Source Table

S.NO	First Name	Last Name
1	Shashank	Sharma
2	Arun	Kumar
3	Shyam	Shankar
4	Sutanu	Kandar

Output Table

S.NO	First Name	Last Name
1	Shashank	Sharma
2	Arun	Kumar
3	Shyam	Shankar

Functions

Basic aggregate functions that operate on a column and return a single value as result

- Aggregate Functions: Operate against a collection of values but return a single summarizing value
- Examples of Basic Aggregate functions:
  - Sum() → returns the summed value of a numeric column.
    - » SELECT sum (column_name) FROM table_name
  - Count() → returns the number of rows that matches a specified criteria
    - » SELECT count (column_name) FROM table_name
  - Max() → returns the maximum value of the selected column
    - » SELECT max(column_name) FROM table_name
  - Min() → returns the minimum value of the selected column
    - » SELECT min(column_name) FROM table_name
  - Avg() → returns the average value of a numeric column
    - » SELECT avg(column_name) FROM table_name

COALESCE function – Used for treating NULL values in a table

Table_1

Employee	Gender	Marital Status	Rating1	Rating2	Rating3
Shyam	M	Married		8	5
Lekha	F	Unmarried	7	10	
Slyvia	F	Married	8	7	10
Prithvi	M	Married	9		6
Das	M	Unmarried	8	4	9
Babu	M	Married			6
Ganeshan	M	Married	1	7	1

Replacing null values with 0s

```
SELECT Employee, Gender, Marital Status,
CASE WHEN Rating1 is null then 0 else Rating1 end as Rating1,
CASE WHEN Rating2 is null then 0 else Rating2 end as Rating2,
CASE WHEN Rating3 is null then 0 else Rating3 end as Rating3,
…….
FROM Table_1
```

How to treat null values without using CASE statement?

Use COALESCE function to replace the null values in any column

```
SELECT Employee, Gender, Marital Status,
COALESCE(Rating1, 0) ,
COALESCE(Rating2, 0) ,
COALESCE(Rating3, 0) ,
…..
FROM Table_1
```

Table_1

Employee	Gender	Marital Status	Rating1	Rating2	Rating3
Shyam	M	Married	0	8	5
Lekha	F	Unmarried	7	10	0
Slyvia	F	Married	8	7	10
Prithvi	M	Married	9	0	6
Das	M	Unmarried	8	4	9
Babu	M	Married	0	0	6
Ganeshan	M	Married	1	7	1

SUBSTRING is used to extract a particular portion of a string

I want the names of users who have only a yahoo mail account from the database

I will start searching each user and their respective mail ids

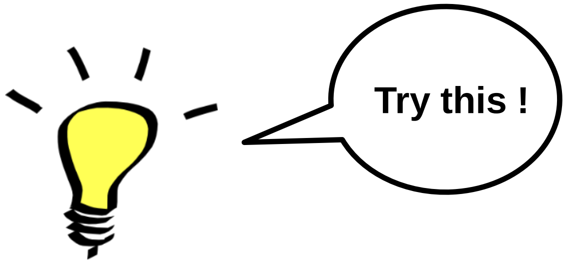
After searching for 999 rows, Hurray! I got the users with yahoo account only

Account_Table

s. no	names	email ids
1	Jhonny Blake	jhonyb@yahoo.com
2	Simi Jain	simi123@gmail.com
3	San roy	sanroy@yahoo.com
4	Jamie khader	jamieK8@yahoo.com
5	Raj James	raj45@gmail.com
6	Abhi Kundra	kabhi78@mu-sigma.com
7	Joy Nil	joynil9@mu-sigma.com
8	Hanu Deys	hanu1@yahoo.com
9	Raju Balan	rajub@yahoo.com
10	Miny Keyer	minyk@gmail.com
11	kimi Jol	jol78@mu-sigma.com
12	Binee sen	bineesen45@gmail.com
.	.	.
.	.	.
.	.	.
998	Neil Dhan	neild@mu-sigma.com
999	Hemu ram	hemr@yahoo.com
1000	Varun vilas	varunv@gmail.com

Did you know, you can do it in an easier way?

- A **substring** function is used to return a portion of a string
- In this case, you can find the substring yahoo from the email ids available in the database and then filter the required ids



```
SELECT S.NO, names, email_ids ,
SUBSTRING(email_ids,charindex(‘@’,email_ids)+1,5) as
accounts
FROM Account_Table
```

Account_Table (1)

S.NO	Names	Email_ids	Accounts
1	Jhonny Blake	jhonyb@yahoo.com	yahoo
2	Simi Jain	simi123@gmail.com	gmail
3	San roy	sanroy@yahoo.com	yahoo
4	Jamie khader	jamieK8@yahoo.com	yahoo
5	Raj James	raj45@gmail.com	gmail
6	Abhi Kundra	kabhi78@mu-sigma.com	mu-si
7	Joy Nil	joynil9@mu-sigma.com	mu-si
8	Hanu Deys	hanu1@yahoo.com	yahoo
9	Raju Balan	rajub@yahoo.com	yahoo
.	.	.	.
.	.	.	.
998	Neil Dhan	neild@mu-sigma.com	mu-si
999	Hemu ram	hemr@yahoo.com	yahoo
1000	Varun vilas	varunv@gmail.com	gmail

Account_Table (2)

S. NO	Names	Email_ids	Accounts
1	Jhonny Blake	jhonyb@yahoo.com	yahoo
3	San roy	sanroy@yahoo.com	yahoo
4	Jamie khader	jamieK8@yahoo.com	yahoo
8	Hanu Deys	hanu1@yahoo.com	yahoo
9	Raju Balan	rajub@yahoo.com	yahoo
.	.	.	.
.	.	.	.
999	Hemu ram	hemr@yahoo.com	yahoo

```
SELECT S.NO, Names, Email_ids,
Accounts FROM Account_table1
 WHERE accounts = ‘yahoo’
```

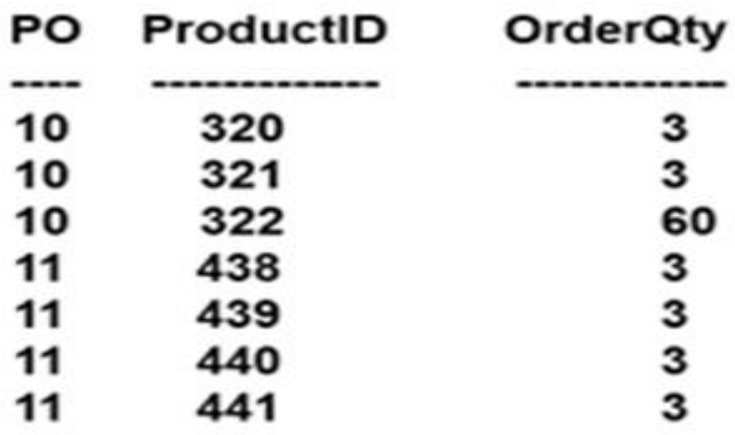
Window functions

Partition By Clause partitions the result of the select statement without changing the level of the data

```
SELECT PO, ProductID, SUM(OrderQty) OVER (PARTITION BY PO ORDER BY ProductID)
FROM TABLE Product_1
```

- Partition By tells SQL to sum(OrderQty) for each different value of column PO

- Order By tells SQL in what order to sum values of OrderQty within every partition of PO



```
PO    ProductID        OrderQty
----  -------------    ------------
10        320                 3
10        321                 3
10        322                 60
11        438                 3
11        439                 3
11        440                 3
11        441                 3
```

Window functions

- A window function is one that can be applied to a partitioned set of rows (known as a *window*) in order to rank or aggregate values in that partition.

- Functions are:
  - **ROW_NUMBER**
  - **RANK**
  - **DENSE_RANK**
  - **NTILE**

Window Functions – Row number

St_id	Name	Subject	Marks
223	Sachin	Maths	98
223	Sachin	English	39
456	Viv	English	24
456	Viv	Maths	98
121	Don	Maths	56
121	Don	Latin	46
456	Viv	Latin	74
121	Don	English	31
223	Sachin	Latin	90

Can I give row numbers for these records??

SELECT *ROW_NUMBER()* OVER (ORDER BY by Marks) as Row_num

Syntax for assigning row number to records

St_id	Name	Subject	Marks	Row_num
223	Sachin	Maths	98	1
456	Viv	Maths	98	2
223	Sachin	Latin	90	3
456	Viv	Latin	74	4
121	Don	Maths	56	5
121	Don	Latin	46	6
223	Sachin	English	39	7
121	Don	English	31	8
456	Viv	English	24	9

Window Functions – Rank

St_id	Name	Subject	Marks
223	Sachin	Maths	98
223	Sachin	English	39
456	Viv	English	24
456	Viv	Maths	98
121	Don	Maths	56
121	Don	Latin	46
456	Viv	Latin	74
121	Don	English	31
223	Sachin	Latin	90

Ok, I have the row numbers. But can I ***rank*** *these records* ***subject – wise****!?*

SELECT *RANK()* OVER (Partition by Subject Order by Marks) as Rank

Syntax for ranking records based on subject wise marks

St_id	Name	Subject	Marks	Rank
223	Sachin	English	39	1
121	Don	English	31	2
456	Viv	English	24	3
223	Sachin	Latin	90	1
456	Viv	Latin	74	2
121	Don	Latin	46	3
223	Sachin	Maths	98	1
456	Viv	Maths	98	1
121	Don	Maths	56	3

Window Functions – Dense Rank

St_id	Name	Subject	Marks
223	Sachin	Maths	98
223	Sachin	English	39
456	Viv	English	24
456	Viv	Maths	98
121	Don	Maths	56
121	Don	Latin	46
456	Viv	Latin	74
121	Don	English	31
223	Sachin	Latin	90

*Can I rank the Math marks of **Don** as 2 rather than 3?*

SELECT *DENSE_RANK()* OVER (Partition by Subject Order by Marks) as Dense_Rank

Syntax for ranking records subject wise without skipping the next rank in case of a tie

St_id	Name	Subject	Marks	Dense_Rank
221	Sachin	English	39	1
121	Don	English	31	2
456	Viv	English	24	3
223	Sachin	Latin	90	1
456	Viv	Latin	74	2
121	Don	Latin	46	3
223	Sachin	Maths	98	1
456	Viv	Maths	98	1
121	Don	Maths	56	2

Window Functions – NTILE

St_id	Name	Subject	Marks
223	Sachin	Maths	98
223	Sachin	English	39
456	Viv	English	24
456	Viv	Maths	98
121	Don	Maths	56
121	Don	Latin	46
456	Viv	Latin	74
121	Don	English	31
223	Sachin	Latin	90

Can I group the whole table into 2 groups?

SELECT *NTILE(2)* OVER (Order by Subject) as Ntile

Syntax for dividing the table into 2 groups

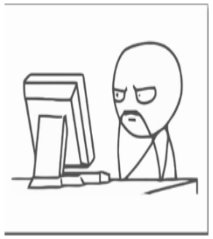
St_id	Name	Subject	Marks	Ntile
223	Sachin	English	39	1
121	Don	English	31	1
456	Viv	English	24	1
223	Sachin	Latin	90	1
456	Viv	Latin	74	1
121	Don	Latin	46	2
223	Sachin	Maths	98	2
456	Viv	Maths	98	2
121	Don	Maths	56	2

Stored Procedures

Stored Procedure is a set of logically group of SQL statement which are grouped to perform a specific task

Initial Code

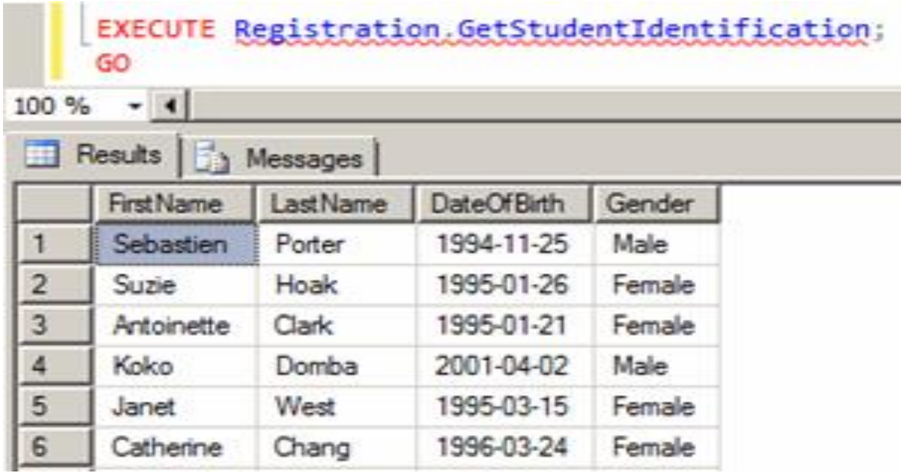
```
select Firstname,Lastname,DateofBirth,Gender
from Registration.Students
.
.
.
select Firstname,Lastname,DateofBirth,Gender
from Registration.Students
.
.
.
select Firstname,Lastname,DateofBirth,Gender
from Registration.Students
```



Can I avoid writing the same line of code again and again?

Creating procedure to get Registration information for all students

```
CREATE PROCEDURE Registration.GetStudentIdentification
AS
BEGIN
   SELECT Firstname, Lastname, DateofBirth, Gender
   FROM Students
END
GO
```



```
EXECUTE Registration.GetStudentIdentification;
GO
```

100 %

Results | Messages

	FirstName	LastName	DateOfBirth	Gender
1	Sebastien	Porter	1994-11-25	Male
2	Suzie	Hoak	1995-01-26	Female
3	Antoinette	Clark	1995-01-21	Female
4	Koko	Domba	2001-04-02	Male
5	Janet	West	1995-03-15	Female
6	Catherine	Chang	1996-03-24	Female

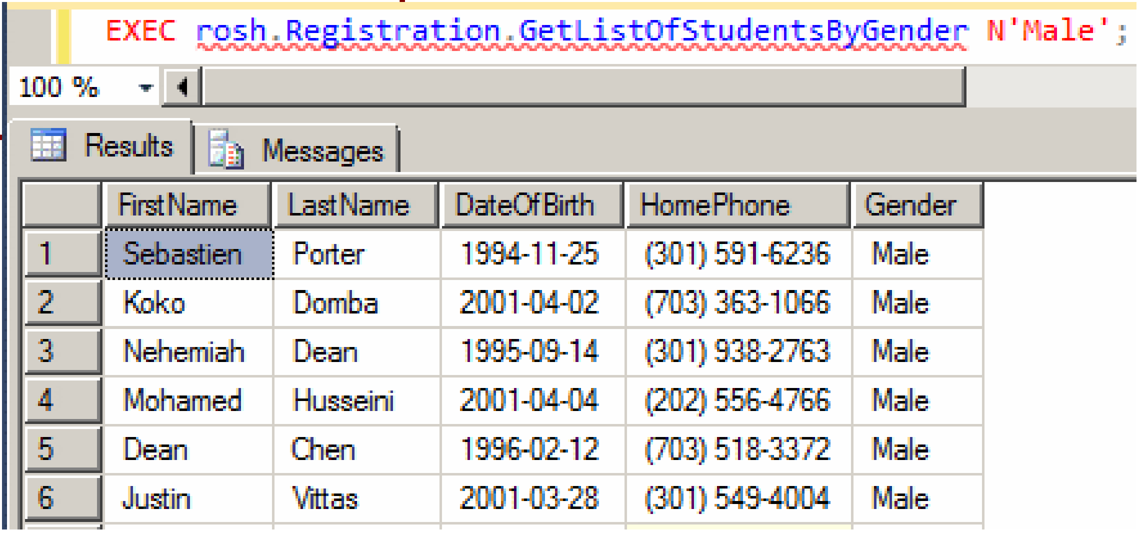
A Stored Procedure can be executed either directly or by passing a parameter value

```
CREATE PROC/PROCEDURE [Schemaname].ProcedureName
@ParameterName  DataType
AS
Body of the Procedure
```

Syntax for creating a stored procedure called by Parameter

```
CREATE PROC/PROCEDURE
Registration.GetListofStudentsByGender
@Gdr NVARCHAR(12)
AS
   SELECT Firstname, Lastname
         DateofBirth, HomePhone, Gender
   FROM Students
   WHERE Gender= @Gdr;
GO
```

Example of a Stored procedure which involves passing a parameter



```
EXEC rosh.Registration.GetListOfStudentsByGender N'Male';
```

	FirstName	LastName	DateOfBirth	HomePhone	Gender
1	Sebastien	Porter	1994-11-25	(301) 591-6236	Male
2	Koko	Domba	2001-04-02	(703) 363-1066	Male
3	Nehemiah	Dean	1995-09-14	(301) 938-2763	Male
4	Mohamed	Husseini	2001-04-04	(202) 556-4766	Male
5	Dean	Chen	1996-02-12	(703) 518-3372	Male
6	Justin	Vittas	2001-03-28	(301) 549-4004	Male

Result after executing the stored procedure

Benefits of Stored Procedures

- **Performance**
  - By grouping SQL statements, a stored procedure allows them to be executed with a single call. This minimizes the use of slow networks, reduces network traffic
- **Productivity and Ease of Use**
  - Avoid redundant coding and increase in productivity
- **Scalability**
  - Stored procedures increase scalability by isolating application processing on the server
- **Security**
  - Stored procedure is helpful in enhancing the security. We can grant permission to the user for executing the Stored procedure instead of giving the permission on the tables used in the Stored procedure

Views

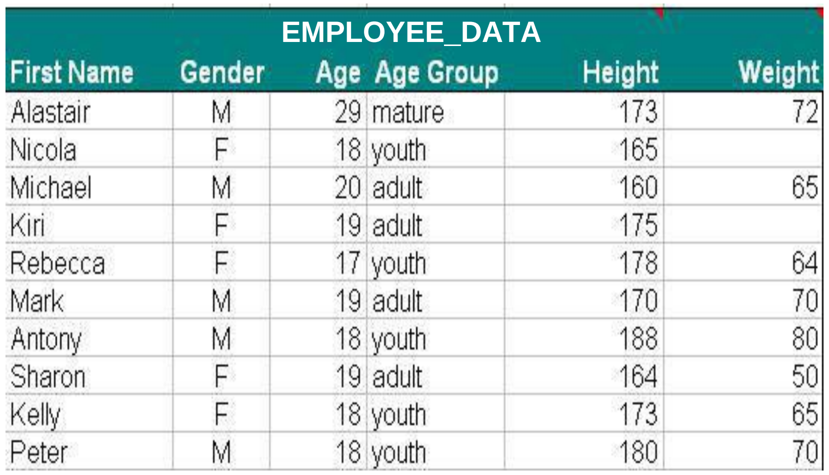
View is a virtual table constructed from a single SQL SELECT statement

How do I let the users access data from EMPLOYEE_DATA without divulging personal details like Height and Weight?

Can create a virtual table, which would allow the user to see only required part of the table but not the original table and simultaneously save space …..

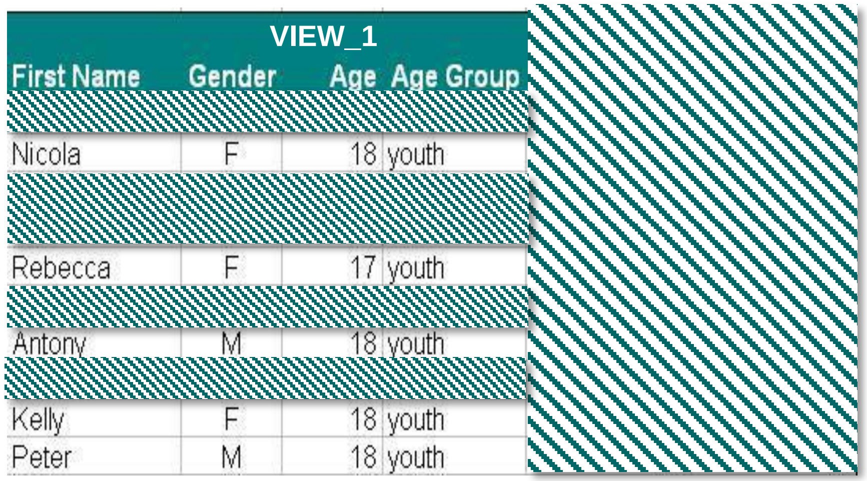
- *Is there any alternative to creating a new table?*
- *Can I save memory space by not creating a table and still enjoy the benefits of table?*

```
CREATE VIEW VIEW_1 AS
SELECT First Name, Gender, Age, Age Group
FROM EMPLOYEE_DATA
WHERE Age Group = ‘youth’
```



EMPLOYEE_DATA

First Name	Gender	Age	Age Group	Height	Weight
Alastair	M	29	mature	173	72
Nicola	F	18	youth	165	
Michael	M	20	adult	160	65
Kiri	F	19	adult	175	
Rebecca	F	17	youth	178	64
Mark	M	19	adult	170	70
Antony	M	18	youth	188	80
Sharon	F	19	adult	164	50
Kelly	F	18	youth	173	65
Peter	M	18	youth	180	70



VIEW_1

First Name	Gender	Age	Age Group
▨	▨	▨	▨
Nicola	F	18	youth
▨	▨	▨	▨
Rebecca	F	17	youth
▨	▨	▨	▨
Antony	M	18	youth
▨	▨	▨	▨
Kelly	F	18	youth
Peter	M	18	youth

View has all the properties of table except that it is a virtual table

- 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

- **Creating a View**

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

```
/*Example*/
CREATE VIEW temp as
SELECT TOP 5 firstname, lastname
FROM dbo.DimCustomer
```

- **Updating a view**

```
ALTER VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

```
/*Example*/
ALTER VIEW temp as
SELECT TOP 5 customerkey
FROM dbo.DimCustomer
```

- **Dropping a View**

```
DROP VIEW view_name
```

```
/*Example*/
DROP VIEW temp
```

Some points on views

It can be created from one or more individual tables

It is a virtual table

Acts as a layer between user and table

It contains data only during runtime. Data gets freed later

VIEWS

Information can be hidden from the user

Memory is saved as a view does not contain any data

Reduces the repetition of queries

Reports can be created

Indexes

Index is a data structure that improves the speed of data retrieval operations on a database table

I want to know the availability of product 990090 in the bunch of products. How do i find it out faster from such a huge table ?

Searching from the first product.

After searching for 900000 rows, Hurray! I found it

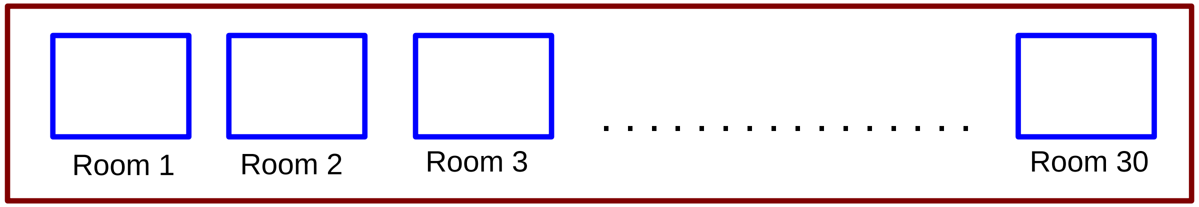
Product_Data

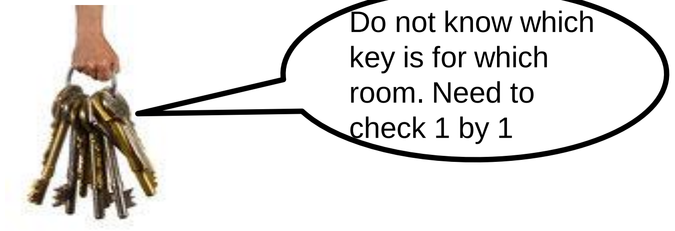
Product_No	Category	Product Description	Availability
200	Games	puzzles	YES
992378	Electronics	chargers	NO
7777	Games	cd's	YES
1001	Basic Needs	soaps	YES
67	Medication	syrups	YES
8999	Games	soft toys	YES
.	.	.	.
.	.	.	.
.	.	.	.
990090	Electronics	mobiles	YES
1000000	Basic Needs	creams	YES

Hello! Did you know that you just wasted your time searching for just one specific product from the bunch of products **?**

Search for a specific object without indexing may waste significant amount of time

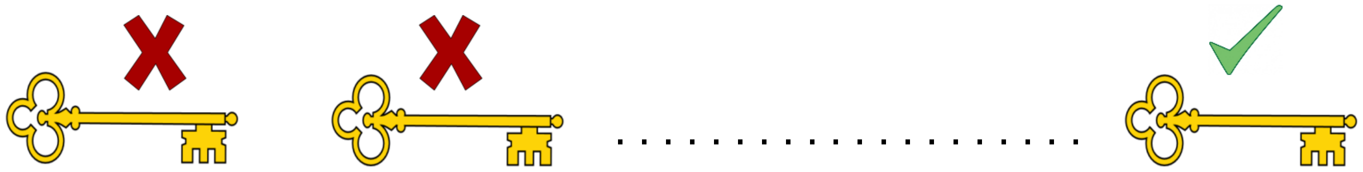
- Consider a set of 30 rooms and 30 keys. Each key can open only 1 room




Do not know which key is for which room. Need to check 1 by 1

Keys for all the rooms

- Now we want to identify the key that can open Room 30

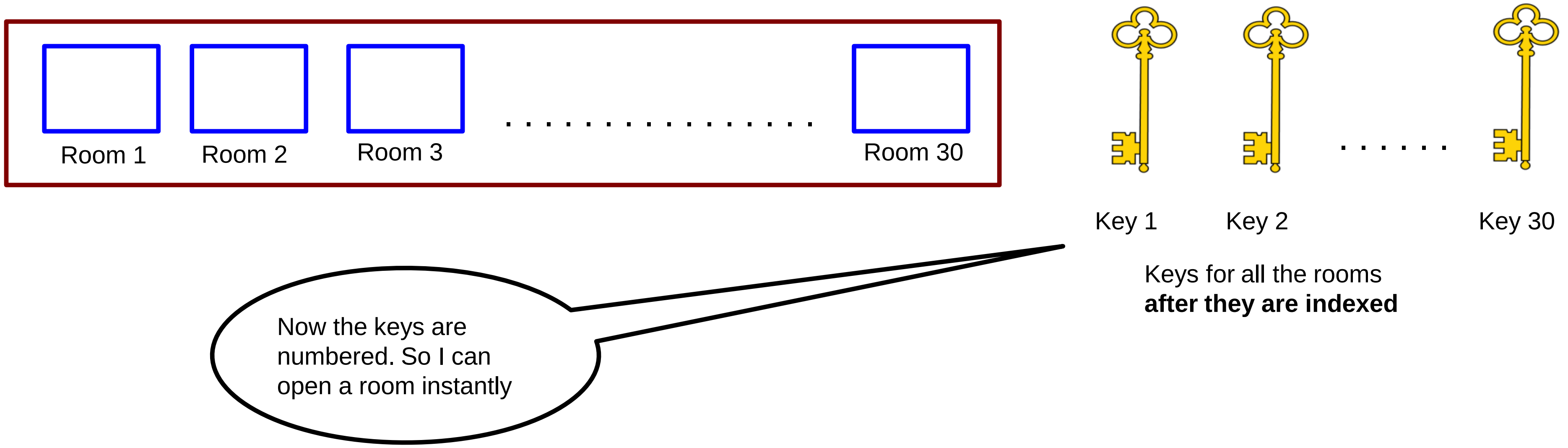


Trial 1 Trial 2 Trial 15

- The correct key for room 30 has been found after 15 iterations. Is it possible to ***decrease the time taken to find the key*** for room 30?

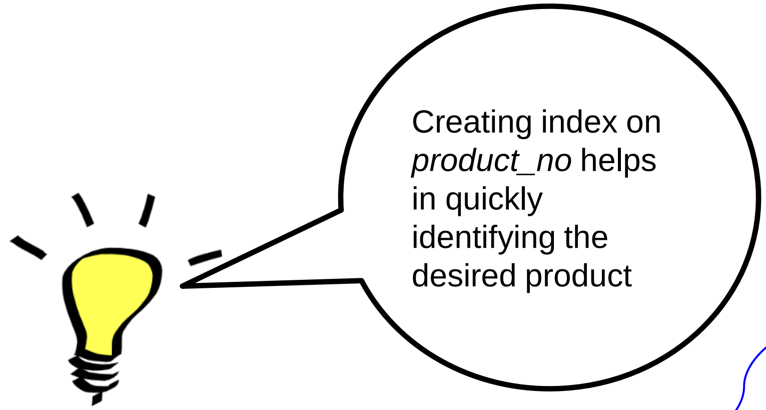
Use of indexing helps increase the speed of search

- Consider the same scenario as previous one,


Room 1
Room 2
Room 3
. . . .
Room 30
Key 1
Key 2
. . . .
Key 30
Keys for all the rooms **after they are indexed**
Now the keys are numbered. So I can open a room instantly

- Now you can see that ***after indexing*** the keys, the ***key for room 30 can be found instantly***!

Creating index on suitable column(s) in a table will reduce the search time significantly


Creating index on *product_no* helps in quickly identifying the desired product

CREATE Index_no on product _data(product _no)

Product_Data

Product_No	Category	Product Description	Availability	Index_no
67	Medication	Syrups	YES	1
140		tablets	No	
500		ointments	YES	
.		.	.	
.		.	.	
1001	Basic Needs	soaps	YES	2
1025		creams	YES	
1777		detergents	YES	
.		.	.	
.		.	.	
.	.	.	.	.
.	.	.	.	.
992378	Electronics	chargers	No	10000
990090		mobiles	YES	
998000		cables	YES	
.		.	.	
.		.	.	

- An ‘Index’ can be created in a table to find data more quickly and efficiently
- Indexes can be created using one or more columns of a database table

Syntax related to Index

1. Creating an index :

```
CREATE INDEX index_name on table_name (column_name)
```

2. Creating an unique index :

```
CREATE UNIQUE INDEX index_name on table_name (column_name)
```

3. Creating an index on a combination of columns

```
CREATE INDEX index_name on table name(col-1,col-2…,col-n)
```

Note:

1. Create indexes on columns that will be frequently searched against
2. The syntax differs amongst different databases . So, check the syntax for creating indexes in your databases

Appendix

Important functions

- Sum() → returns the summed value of a numeric column.
  - SELECT sum (column_name) FROM table_name
- Count() → returns the number of rows that matches a specified criteria
  - SELECT count (column_name) FROM table_name
- Max() → returns the maximum value of the selected column
  - SELECT max(column_name) FROM table_name
- Min() → returns the minimum value of the selected column
  - SELECT min(column_name) FROM table_name
- Avg() → returns the average value of a numeric column
  - SELECT avg(column_name) FROM table_name
- Lower() → converts the value of a field to lower case
  - SELECT lower(col_name) from table
- Upper() → converts the value of a field to upper case
  - SELECT upper(col_name) from table

Numeric functions

- Abs(X) – gives the absolute value Eg: SELECT ABS(-2); → 2
- Ceiling (X) – returns the smallest integer not less than x Eg: SELECT CEILING(1.23); → 2
- Floor (X) – returns the largest value not greater than X Eg: SELECT FLOOR(1.23); → 1
- Mod(n,m) – remainder of n divided by m Eg: SELECT MOD(234, 10); → 4
- Pow(x,y) – x to the power y
- Round(x),round(x,d) – rounds off x (to d decimal places) Eg: SELECT ROUND(1.298, 1); → 1.3
- Sign(X) – gives the sign of the value – positive/negative (-1,0,+1)
- Truncate(x,d) – truncates x after d decimal points Eg: SELECT TRUNCATE(1.999,0); → 1 if d is negative then the d number of places before the decimal point becomes 0
- div – used as a math operator. Divides and then floors the value. Eg; SELECT 5 div 2 → Here, the result is '2'

String Functions

- ltrim(string,trimlist) rtrim(string,trimlist) → returns string with the leftmost/rightmost char that match the chars in trim list removed. If the trimlist is not mentioned then it returns the string with its leading/trailing blanks removed.

- Inticap(string) → coverts the first letter of the string to caps

- Substring(str,pos,len) → returns substring of string which begins at pos and is len characters long . Synonym for mid()
  - SELECT substr('Quadratically',5); →'ratically'

- Charindex(‘<character>’, <string>) → Returns the position(interger) of a character in a string

- Length(string) → length of the string (same as len())

- replace(string,target,replacement) → returns string with all occurrences of target replaced with replacement

String Functions contd…

- Ascii(str) – returns the ASCII value of the character

- Char(<integers with or without a quotes>) – returns the character corresponding to the ascii integer

- Insert(str,pos,len,newstr) – returns str with newstr substituted at the pos mentioned with len characters long
  - SELECT INSERT('Quadratic', 3, 4, 'What'); → 'QuWhattic'

- Repeat(str,count) – repeats a particular string 'count' number of times(similar to replicate)
  - SELECT REPEAT('MySQL', 3); → 'MySQLMySQLMySQL'

- Reverse (str) – returns the reverse of the string Eg: SELECT reverse(col_name) from table

- Locate (substring,str) – returns position of where the substring starts in the string
  - SELECT LOCATE('bar', 'foobarbar'); → 4

Date functions

- now() → returns the current system date and time(similar to curdate(),sysdate()
- date(date_time_expression) →gives the date part of the date time expression
- datediff () → gives the number of days between 2 dates given as parameters
- day() / dayofmonth() → gives the day of the date in the month (1-31)
- dayofweek() → gives the day of the date in the week (1-7)(1=Sunday,2=Monday….)
- dayofyear() → gives the day of the date in the year (1-366)
- timediff(date_time1,date_time2) – gives the time difference between the 2 values
- hour() , minute (), second() – gives corresponding value from the time
- month(), year () - gives corresponding value from the date
- dayname(), monthname() – gives the name of the day and the month from the date

Date functions contd…

- Weekofyear() – gives the week of the year (1-53)
- Quarter() – gives a value from 1-4 in which the date lies
- Addtime (expr1,expr2) – adds expr2 to expr1, where expr2→time expression, expr1→time or datetime expression
- Adddate (date,INTERVAL expr unit) – adds the expr to the date depending on the unit(day,month,year) similar to date_add()
  - SELECT ADDDATE('2008-01-02',INTERVAL 31 DAY) Results : 2008-02-02
- date_ format() → formats the display of the date/time in the specified format
  - Select date_format ('datetime1','%a %b ….')
  - %W – day of the week
  - %M – month
  - %Y – year
  - %H – hour (24hr format)
  - %i – minutes
  - %s – seconds
  - %r – 12 hour format

Misc. functions

- First() → returns the first value of the selected column
  - Select first(col_name) from table
- Last() → returns the last value of the selected column
  - Select last(col_name) from table
- Trim(‘string’) – removes spaces at the ends of the string
- Coalesce (x,y,z) – checks if values in order are not null and returns it. Same as ifnull(x,y,z)
- Cast (expr as type[(length)]) – takes expr of one type and produces a value of the type mentioned
- Limit <number> - specified at the end, it limits the number of observations
- Top – select top <n> * from….., selects the top n rows
- Between – between value1 and value2 – checks if the value falls between the 2 mentioned values