Relational Databases

Select Cheat Sheet

Phases

```
❺ SELECT empid, YEAR(orderdate) AS orderyear, COUNT(*) AS numorders
❶ FROM Sales.Orders
❷ WHERE custid = 71
❸ GROUP BY empid, YEAR(orderdate)
❹ HAVING COUNT(*) > 1
❻ ORDER BY empid, orderyear;
```

❶ FROM	Specify the table we want to query
❷ WHERE	Uses a predicate to filter the rows returned. Where clauses enable the use of indices to improve performance and reduce the network traffic that would occur if we performed a table scan and filtered on the client.
❸ GROUPBY	Produce a group for each unique combination of values specified in this clause.
❹ HAVING	Uses a predicate to filter the groups returned. Can utilise aggregate functions in the predicate.
❺ SELECT	Specify the columns we want to see in the result
❻ ORDER BY	Sort the rows for presentation purposes

GROUP BY

If a query contains a group by phase any subsequent HAVING, SELECT, and ORDERBY clauses work on groups. As such they can only operate on expressions that return a single scalar value per group. Any fields specified in the GROUP BY phase implicitly have this process. Any elements that do not meet this restriction can only be used as inputs to aggregation functions such as COUNT, SUM, AVG, MIN, MAX.

ORDER BY

In SQL, a table has no order. By using order by the result is ordered and hence cannot be considered a table. In SQL the ordered rows are referred to as a cursor. Unlike all other phases the order by phase can utilise column alisases defined in the select phase as the order by is the only phase that follows the select phase.

Phase Illustrations

SELECT

```
❷ SELECT *
❶ FROM TelNumber
```

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
3	2	01324256123
4	2	07999321145
5	NULL	02074257777

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
3	2	01324256123
4	2	07999321145
5	NULL	02074257777

WHERE?

Where add a predicate to filter the results.

```
❸ Select *
❶ FROM TelNumber
❷ WHERE personId IN (1,2)
```

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
4	2	07999321145
5	NULL	02074257777

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
4	2	07999321145

GROUP BY

Allows grouping. The select can only work on columns appropriate to the grouping.

```
❹ Select personId, COUNT(telNumber) AS 'Number Count'
❶ FROM TelNumber
❷ WHERE personId IN (1,2)
❸ GROUP BY personId
```

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
4	2	07999321145
5	NULL	02074257777

personId	Number Count
1	2
2	1

HAVING

```
❺ Select personId, COUNT(telNumber) AS 'Number Count'
❶ FROM TelNumber
❷ WHERE personId IN (1,2)
❸ GROUP BY personId
❹ HAVING count(telNumber) > 1
```

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
4	2	07999321145
5	NULL	02074257777

personId	Number Count
1	2

Examples

Basic Select

The following queries work on the below Person table.

id	firstName	secondName
1	Kenneth	Wilson
2	John	Smith
3	Kelly	Clarkson

Name **Code**

Select

```
SELECT *
FROM Person
```

id	firstName	secondName
1	Kenneth	Wilson
2	John	Smith
3	Kelly	Clarkson

Select Named Columns

```
SELECT P.firstName
FROM Person P
```

firstName
Kenneth
John
Kelly

Select AS

```
SELECT P.firstName AS "First"
FROM Person P
```

First
Kenneth
John
Kelly

WHERE (FILTRATION)

The following queries work on the below Person table.

id	firstName	secondName
1	Kenneth	Wilson
2	John	Smith
3	Kelly	Clarkson
3	John	Wilson

Name **Code**

Equality

```
SELECT firstName
FROM Person P
WHERE firstName = 'John'
```

firstName
John
John

Inequality

```
SELECT *
FROM Person
WHERE firstName <> 'John'
```

id	firstName	secondName
1	Kenneth	Wilson
3	Kelly	Clarkson

Like

```
SELECT *
FROM Person
WHERE firstName LIKE '%enn%'
```

id	firstName	secondName
1	Kenneth	Wilson

IN

```
SELECT *
FROM Person
WHERE firstName

    IN ('Kenneth', 'John')
```

id	firstName	secondName
1	Kenneth	Wilson
2	John	Smith
3	John	Wilson

NOT

```
SELECT *
FROM Person
WHERE firstName
  NOT IN ('Kenneth', 'John')
```

id	firstName	secondName
3	Kelly	Clarkson

OR

```
SELECT *
FROM Person
WHERE firstName = 'John'
    OR firstName = 'Kelly'
```

id	firstName	secondName
2	John	Smith
3	Kelly	Clarkson
3	John	Wilson

AND

```
SELECT *
FROM Person
WHERE firstName = 'John'
    AND secondName = 'Smith'
```

id	firstName	secondName
2	John	Smith

The following queries work on the below Products table.

productId	description
1	European Call
2	Variance Swap

Name	Code	
Less than	`SELECT * FROM Product WHERE productId < 2`	productId: 1, description: European Call
Less than or equal to	`SELECT * FROM Product WHERE productId <= 2`	productId: 1, description: European Call; productId: 2, description: Variance Swap
Greater than	`SELECT * FROM Product WHERE productId > 1`	productId: 2, description: Variance Swap

Less than

```
SELECT *
FROM Product
WHERE productId < 2
```

productId	description
1	European Call

Less than or equal to

```
SELECT *
FROM Product
WHERE productId <= 2
```

productId	description
1	European Call
2	Variance Swap

Greater than

```
SELECT *
FROM Product
WHERE productId > 1
```

productId	description
2	Variance Swap

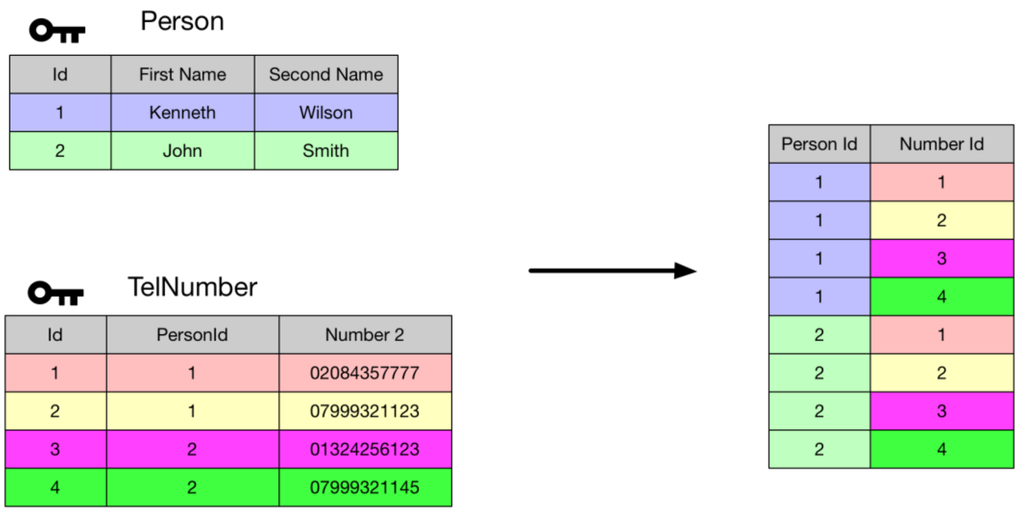
Insert Cheat Sheet

```
INSERT INTO Person
     (firstname, secondname, age, countryId)
VALUES
     ('Indiana', 'Jones', 50, 3),
     ('Carl',    'Jones', 50, 3)
```

Join Cheat Sheet

Cross Join

```
SELECT p.id AS 'Person Id', T.id AS 'Number Id'
FROM Person AS P
    CROSS JOIN TelNumber as T
```



Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith

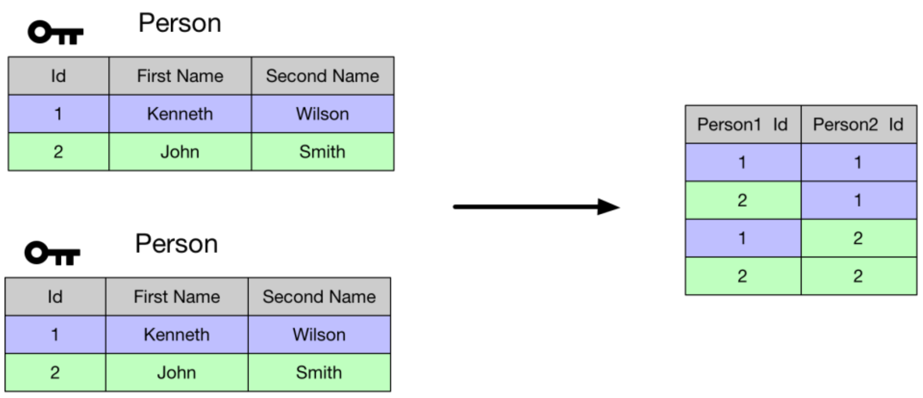
TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
3	2	01324256123
4	2	07999321145

Person Id	Number Id
1	1
1	2
1	3
1	4
2	1
2	2
2	3
2	4

Self-Cross Join

```
SELECT P1.id AS 'Person1 Id', P2.id AS 'Person2 Id'
FROM Person AS P1
    CROSS JOIN Person AS P2
```



Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith

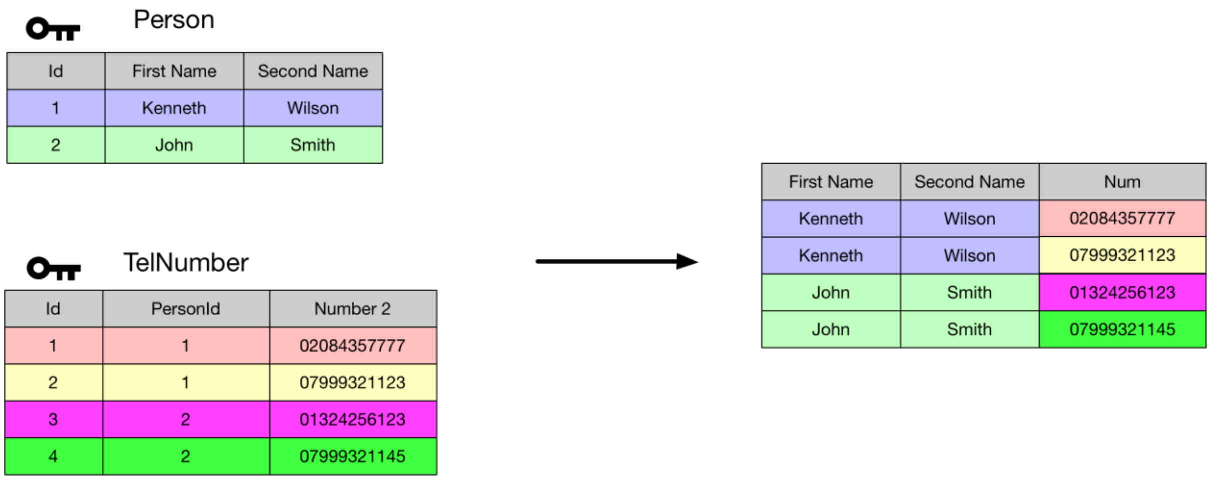
Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith

Person1 Id	Person2 Id
1	1
2	1
1	2
2	2

Inner Join

```
SELECT
    P.firstName AS 'First Name',
    P.secondName AS 'Second Name',
    T.telNumber AS 'Num'
FROM Person AS P
INNER JOIN TelNumber AS T
    ON P.id = T.personId
```



Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith

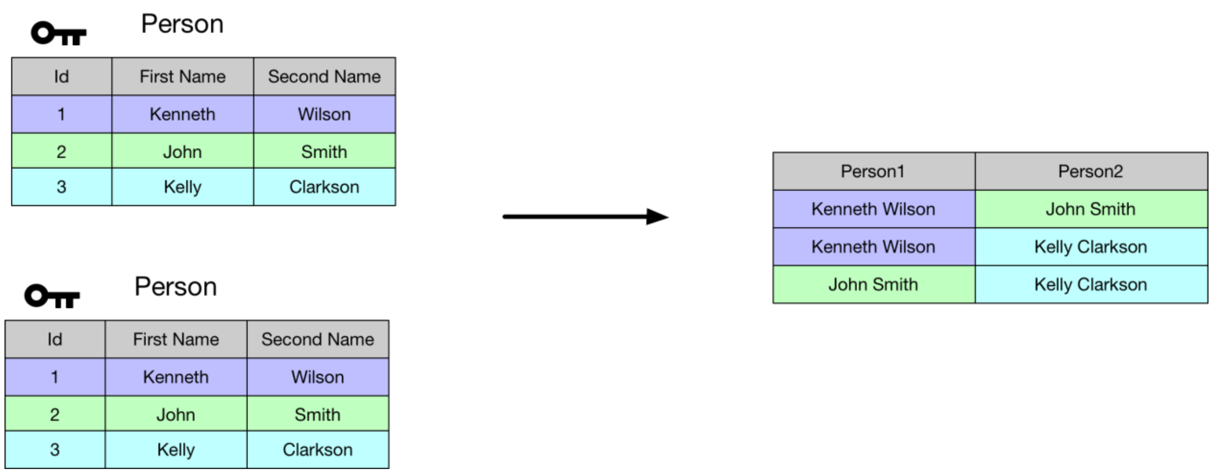
TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
3	2	01324256123
4	2	07999321145

First Name	Second Name	Num
Kenneth	Wilson	02084357777
Kenneth	Wilson	07999321123
John	Smith	01324256123
John	Smith	07999321145

Non Equi-Join

```
SELECT
    P1.firstName + ' ' + P1.secondName AS 'Person1',
    P2.firstName + ' ' + P2.secondName AS 'Person2'
FROM Person AS P1
    INNER JOIN Person AS P2
          ON P1.id < P2.id
```



Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith
3	Kelly	Clarkson

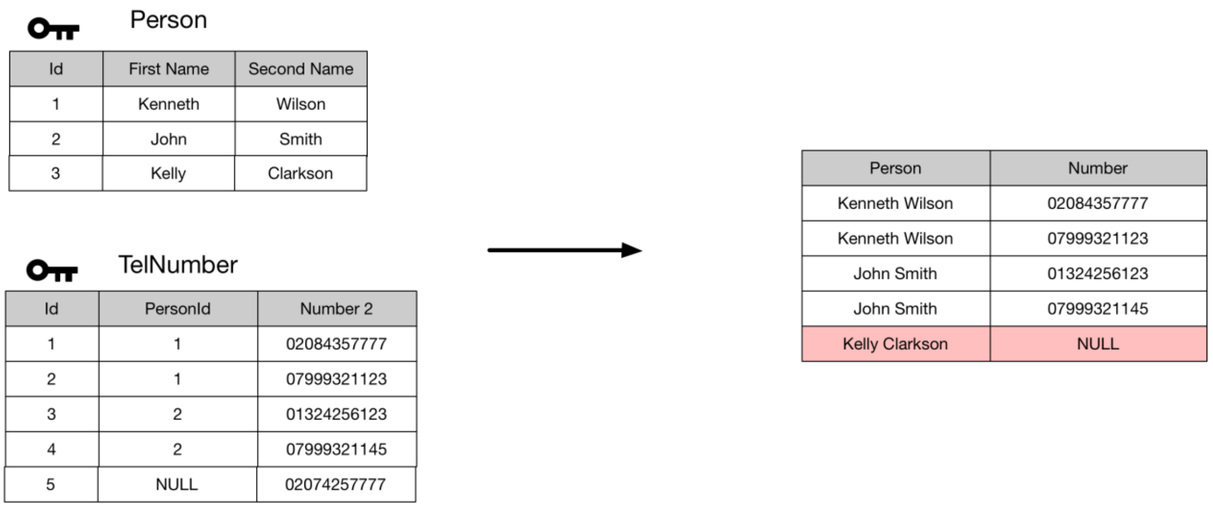
Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith
3	Kelly	Clarkson

Person1	Person2
Kenneth Wilson	John Smith
Kenneth Wilson	Kelly Clarkson
John Smith	Kelly Clarkson

Left Outer Join

```
SELECT
    P.firstName + ' ' + P.secondName AS 'Person',
    T.telNumber AS 'Number'
FROM Person AS P
    LEFT OUTER JOIN TelNumber AS T
          On P.id = T.personId
```



Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith
3	Kelly	Clarkson

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
3	2	01324256123
4	2	07999321145
5	NULL	02074257777

Person	Number
Kenneth Wilson	02084357777
Kenneth Wilson	07999321123
John Smith	01324256123
John Smith	07999321145
Kelly Clarkson	NULL

Right Outer Join

```
SELECT
    P.firstName + ' ' + P.secondName AS 'Person',
    T.telNumber AS 'Number'
FROM Person AS P
    RIGHT OUTER JOIN TelNumber AS T
          On P.id = T.personId
```

Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith
3	Kelly	Clarkson

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
3	2	01324256123
4	2	07999321145
5	NULL	02074257777

Person	Number
Kenneth Wilson	02084357777
Kenneth Wilson	07999321123
John Smith	01324256123
John Smith	07999321145
NULL	02074257777

Normalization Cheat Sheet

Forms

1st Normal Form

For a table to be in first normal form each cell must contain only one value from the domain. The following table violates this restriction

Id	First Name	Second Name	Numbers
1	Kenneth	Wilson	02084357777, 07999321123
2	John	Smith	01324256123, 07999321145

We should note the following table is still in violation of 1st normal form as repeating columns groups are also disallowed.

Id	First Name	Second Name	Number 1	Number 2
1	Kenneth	Wilson	02084357777	07999321123
2	John	Smith	01324256123	07999321145

We can fix this using a schema something like the following. The PersonId field acts as the foreign key that indexes into the person table

Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
3	2	01324256123
4	2	07999321145

The second restriction on 1st normal form is that every row is unique. We can ensure uniqueness of rows by applying a candidate key to the row.

Person

Id	First Name	Second Name
1	Kenneth	Wilson
2	John	Smith

TelNumber

Id	PersonId	Number 2
1	1	02084357777
2	1	07999321123
3	2	01324256123
4	2	07999321145

2nd Normal Form

Second normal for applies to relations with composite keys. Where there is a composite key, we should not be able to locate the value of any non-key attribute using only part of the composite key. The following table is in violation of 2nd normal form.

Make	Model	Manufacturer Country
Ford	Fiesta	US
Ford	Focus	US

We can fix this as follows

Make	Model
Ford	Fiesta
Ford	Focus

Make	Manufacturer Country
Ford	US

A relation with a single attribute primary key in 1st normal form is automatically in 2nd normal form.

3rd Normal Form

To be in third normal form the relation must first be in second normal form. The second rule is that no non-key attribute can be identified by another non-key attribute. This table is in violation of 3rd normal form

PosId	ProductId	ProductType
1	2	Derivative
2	2	Derivative

We can fix this as follows.

PosId	ProductId
1	2
2	2

ProductId	ProductType
2	Derivative
2	Derivative

We can summarise the second and third forms as meaning than in order to identify the value of any non-key attribute we need to use the full primary key. Furthermore we cannot identify the value of any non-key field using another non-key field.

Overview

An RDBMS is supposed to implement the relational model and provide the means to store, manage, enforce the integrity of, and query data.

- T-SQL Fundamentals

The relational model is based on set theory and predicate logic. SQL is a declarative language in that we describe what we want done and leave the details to the RDBMS. A relation in SQL consists of a heading that specifies the set of attributes (columns) and a body that consists of a set of tuples (rows).

Constraints

RDBMS allows one to model data integrity by specifying constraints. A candidate key is specified on an attribute (column) to enforce uniqueness of tuples (rows). One of the candidate keys is chosen as the primary key and forms the preferred way of uniquely identifying rows. Foreign key is specified on a referencing relation. A foreign key constraint can be used to enforce referential integrity by ensuring only values that exist in the referenced relations are allowed in the referencing relations foreign key.

If we apply key constraints to a table, each element is unique, and it can be considered a set (Otherwise it is a bag or a multiset). Order is unimportant in a set. For this reason, the result of a query has no order unless we explicitly give some sorting criteria.

Predicates

A predicate is an expression that is either true or false. Predicates can be used to

1. Enforce data integrity
2. Filter data into subsets.
3. Specify sets by their properties rather than explicit enumeration of elements

All SQL commands can be split into three buckets.

	Column Header
Data Definition Language	CREATE , ALTER, DROP
Data Manipulation Language	SELECT, INSERT, UPDATE, DELETE, TRUNCATE, MERGE
Data Control Language	GRANT and REVOKE

Query Language

Predicates

IN

Returns true if a value or scalar expression is contained in a specified set of values.

```
SELECT * from Products
WHERE prodId IN (1, 2, 3)
```

BETWEEN

```
SELECT * from Products
WHERE prodId BETWEEN 1 AND 3
```

LIKE

```
SELECT * from Products
WHERE description LIKE 'E%'
```

Operators

Equality and Ordinality

Operator	Description
=	Equality
>	
<	
>=	
<=	
<>	

Logical

Operator	Description
AND	and
OR	
NOT	

Arithmetic

Operator	Description
+	
-	
*	
/	
%	Modulo

If two arguments are of the same type the result is of the same type. So, 7 / 2 = 3. We might want to perform a cast in this instance as follows.

```
SELECT CAST(7 as Numeric(12,2))/ CAST(3 as Numeric(12,2)) as num
```

Three Valued Logic

In SQL predicates can evaluate to `TRUE`, `FALSE` or `UNKNOWN`. If one of the arguments in a logical expression is NULL, then the result is `UNKNOWN`. If a logical expression is used in a query filter, then any result of UNKNOWN leads to a rejection (accept true). If a logical expression used in a check constraint returns UNKNOWN, the value is accepted (reject false)

Type

Characters

Characters can be either Unicode or regular

Type	Description
CHAR	Fixed length, 1 byte per character
VARCHAR	
NCHAR	Fixed length Unicode, 2 bytes per character
NVARCHAR	

Date and Time

Type	Description
DATETIME	Legacy
SMALLDATETIME	Legacy
DATE	3 bytes representing range from 00010101 to 99991231
TIME	3 to 5 bytes giving accuracy to 100 nanoseconds
DATETIME2	6 to 8 bytes giving date and time accuracy to same level as DATE and TIME combined
DATETIMEOFFSET	8 to 10 bytes. Similar to DATETIE2 but includes offset from UTC

To create date and time values when use string literals which are then implicitly converted to the relevant type. When using literals the best practice is to use the language neutral format ‘YYYYMMDD’. (Other formats depends on the language of the session)

```
insert Trades values(2,'20200103')
select tradeId AS 'Trade Id', tradeDate AS 'Trade Date' from Trades
```

Trade Id	Trade Date
2	03/01/2020 00:00:00

> **LEGACY DATETIME TYPE**
>
> When dealing with the legacy type DATETIME the convention is that we use a time of midnight if we want to only use the date part and a date of January 1st 1900 if we are only interested in the time part.

INEFFICIENT OF MANIPULATION COLUMNS IN FILTERS

We need to careful when applying filters such as WHERE phases. If we manipulate the filtered column this can prevent the database server using the index in an efficient manner. The following is an inefficient query

```
SELECT tradeId AS 'Trade Id', tradeDate AS 'Trade Date'
FROM Trades
WHERE YEAR(tradeDate) = 2020
```

We can improve our query using a range filter as follows which enables the efficient use on indices.

```
SELECT tradeId AS 'Trade Id', tradeDate AS 'Trade Date'
FROM Trades
WHERE tradeDate >= '20200101' AND tradeDate < '20210101'
```

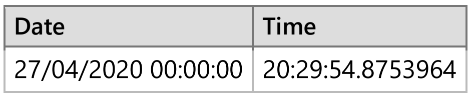
FUNCTIONS

FUNCTIONS

Function	Returns	Description
GETDATE	DATETIME	Gets current date and time
GETUTCDATE	DATETIME	Get current date and time in UTC
SYSDATETIMEOFFSET	DATETIMEOFFSET	Get current date and time with UTC offset
SWITCHOFFSET		Switches to a different UTC Offset (Timezone)
DATEADD		Add years, months or days to a
DATEDIFF		Give different between dates in some date part (year, month, day etc)
DATEPART		Get year, month, day etc
YEAR,MONTH,DAY		Abbreviations for DATEPART

As none of the above functions return only the date or only the time we need to do a little extra to get these

```
SELECT
    CAST(SYSDATETIME() AS DATE) AS [Date],
    CAST(SYSDATETIME() AS TIME) AS [Time]
```



Date	Time
27/04/2020 00:00:00	20:29:54.8753964

MetaData

Architecture

```
A single SQL server can hold multiple user databases in addition to a set
of system databases (tempdn, model etc)
```

Data Definition

The following creates a table.

```
DROP TABLE IF EXISTS dbo.Products;

CREATE TABLE dbo.Products
(
    prodId INT NOT NULL,
    description VARCHAR(30) NOT NULL
);
```

We can setup declarative data integrity. First, we show how to add a primary key constraint.

Primary Key Constaint

A primary key has the following properties

- Each table can have only one primary key
- The fields making up the primary key cannot be null
- The server creates an index to efficiently enforce uniqueness and retrieval

```
ALTER TABLE dbo.Products
    ADD CONSTRAINT PK_Products
    PRIMARY KEY(prodId);
```

Foreign Key Constraint

SQLServer Install

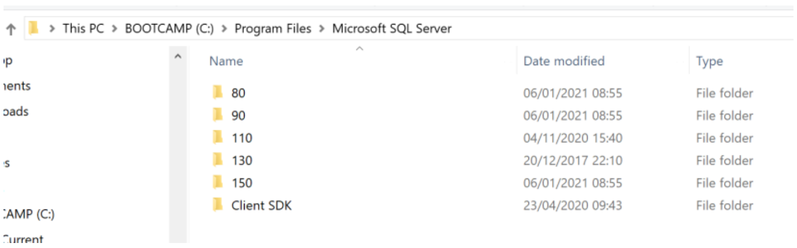
LocalDB

Install

Use the following link and choose the third one from the top

https://www.hanselman.com/blog/download-sql-server-express

The default locations of the installs are `C:\Program Files\Microsoft SQL Server` On my machine I see multiple versions.



This PC > BOOTCAMP (C:) > Program Files > Microsoft SQL Server

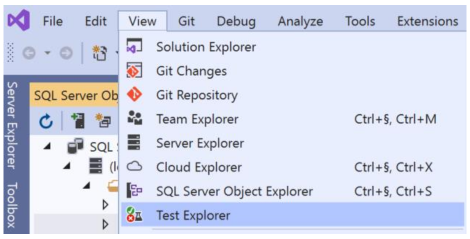
Name	Date modified	Type
80	06/01/2021 08:55	File folder
90	06/01/2021 08:55	File folder
110	04/11/2020 15:40	File folder
130	20/12/2017 22:10	File folder
150	06/01/2021 08:55	File folder
Client SDK	23/04/2020 09:43	File folder

Start/Manage

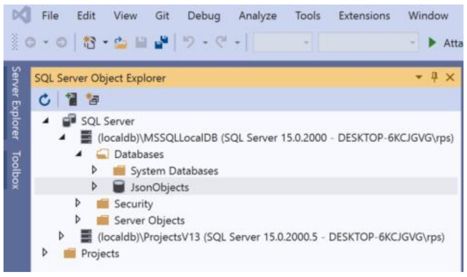
LocalDb supports two kinds on instance: Automatic instances and named instances. The automatic instance for SQLServer is called MSSQLLocalDb.

Visual Studio

The easiest way to manage the local DB instances is to use visual studio.



And when open it looks like this.



```
SQL Server
  (localdb)\MSSQLLocalDB (SQL Server 15.0.2000 - DESKTOP-6KCJGVG\rps)
    Databases
      System Databases
      JsonObjects
    Security
    Server Objects
  (localdb)\ProjectsV13 (SQL Server 15.0.2000.5 - DESKTOP-6KCJGVG\rps)
Projects
```

Command Prompt

Open a command prompt. We can list all instances of SQLServer using the command.

```
C:\Users\rps>"C:\Program Files\Microsoft SQL
Server\150\Tools\Binn\SqlLocalDB.exe" info

>> MSSQLLocalDB
```

We create a named instance as.

```
C:\Users\rps>"C:\Program Files\Microsoft SQL
Server\150\Tools\Binn\SqlLocalDB.exe" create KennysLocalDb
```

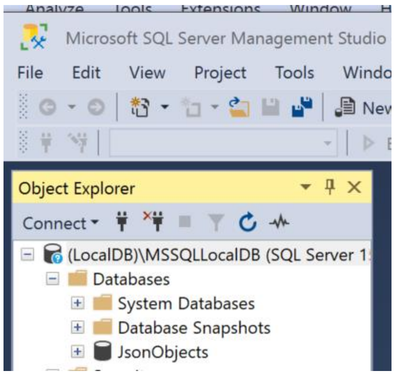
We can view a named install.

```
"C:\Program Files\Microsoft SQL Server\150\Tools\Binn\SqlLocalDB.exe"
info KennysLocalDb

>> Name:                 KennysLocalDb
>> Version:              15.0.2000.5
>> Shared name:
>> Owner:                DESKTOP-6KCJGVG\rps
>> Auto-create:          No
>> State:                Stopped
>> Last start time:      19/01/2021 17:47:37
>> Instance pipe name:
```

Microsoft SQL Server Management Studio

If we open management studio it should just automatically connect to the local DB.

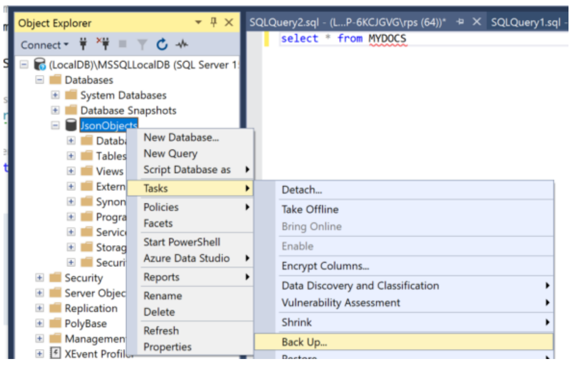


Let us add a JSON table.

```
create table MYDOCS (
    ID bigint primary key identity,
    DOC nvarchar(max)
);
```

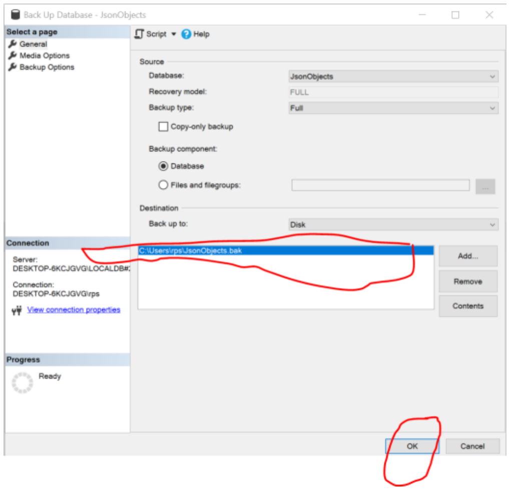
SQLServer Backup/Restore

The easiest way to backup and restore our database is to use Microsoft SQL Server Management Studio. Right click the database in the Object Explorer and select as follows.



```
select * from MYDOCS
```

You will then see the following. Just click ok which backs it up to the current user's home directory


Back Up Database - JsonObjects

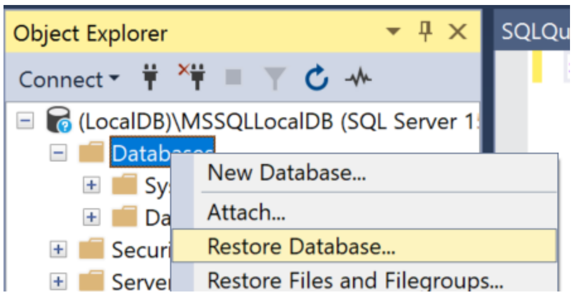
Source
Database: JsonObjects
Recovery model: FULL
Backup type: Full
Copy-only backup: ☐
Backup component: [✓ Database] Files and filegroups:

Destination
Back up to: Disk
C:\Users\rps\JsonObjects.bak

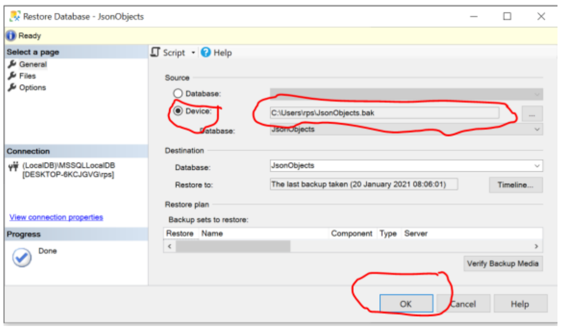
Connection
Server: DESKTOP-6KCJGVG\LOCALDB#
Connection: DESKTOP-6KCJGVG\rps

Progress
Ready

We can now safely delete our database safe in the knowledge we can back it up. One deleted restore it as follows. In the Object Explorer right-click on `databases` and select `Restore Database`.



No in the dialogue select Device and enter the location of the file



Source

Database:

[✓ Device:] C:\Users\rps\JsonObjects.bak

Database: JsonObjects

Destination

Database: JsonObjects

Restore to: The last backup taken (20 January 2021 08:06:01)

Restore plan

Backup sets to restore:

Restore	Name	Component	Type	Server

Connection: (LocalDB)\MSSQLLocalDB [DESKTOP-6KCJGVG\rps]

SQLServer Docker

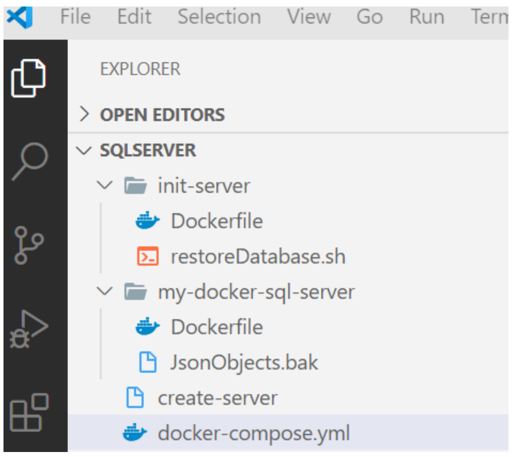
We can run SQLServer 2019 inside docker. The images are here.

Run SQL Server Instance

See

Create and Populate Database.

The following docker logic shows how to get an instance running and copy in a backup file to restore a database and its objects. Note that for the following example we already need backup file taking from somewhere that we can use to do the restore. Our structure looks like this.



```
SQLSERVER
  init-server
    Dockerfile
    restoreDatabase.sh
  my-docker-sql-server
    Dockerfile
    JsonObjects.bak
  create-server
  docker-compose.yml
```

Listing 1 my-docker-sql-server/Dockerfile

```
FROM mcr.microsoft.com/mssql/server:2019-latest
USER root
COPY JsonObjects.bak /var/opt/mssql/data/JsonObjects.bak
CMD /opt/mssql/bin/sqlservr
```

Listing 2 init-server/Dockerfile

```
FROM mcr.microsoft.com/mssql-tools
USER root
COPY restoreDatabase.sh /restoreDatabase.sh
CMD "./restoreDatabase.sh"
```

Listing 3 init-server/restoreDatabase.sh

```
echo "sleeping to allow db to start up"
sleep 10
echo 'resoting database JsonObject'
/opt/mssql-tools/bin/sqlcmd -S my-docker-sql-server -U 'SA' -P $SA_PASSWORD -
Q "RESTORE DATABASE [JsonObjects] FROM DISK = N'/var/opt/mssql/data/JsonObjects.bak' WITH R
EPLACE"
```

NOTE: FILE ENDING

For the above shell script make sure the line ending in VS code is set to LF or it will not work.

Listing 4 docker-compose.yml

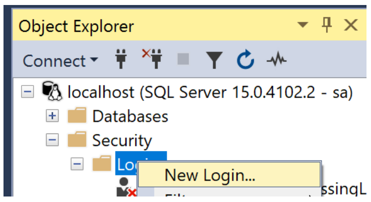
```
version: '3'
services:
  init-server:
    build:
      context: init-server
      dockerfile: Dockerfile
    environment:
      SA_PASSWORD: "Pa!ssWordTwo!"
    links:
      - my-docker-sql-server
  my-docker-sql-server:
    build:
      context: my-docker-sql-server
      dockerfile: Dockerfile
    environment:
      SA_PASSWORD: "Pa!ssWordTwo!"
      ACCEPT_EULA: "Y"
    ports:
      - 1433:1433
```

Logins, Users and Roles

A login is a server level entity and users are database level entities. We can have a login with no user associated. In this case we can log onto the server but not use any of the databases on the server.

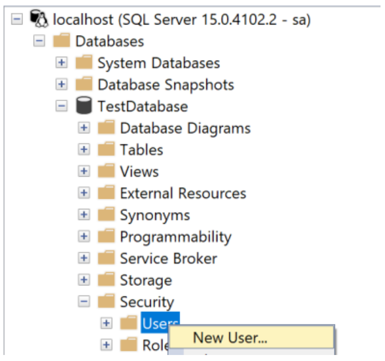
Create Login

On the folder `<ServerName>/Security/Logins` right click and select New Login.



Create User

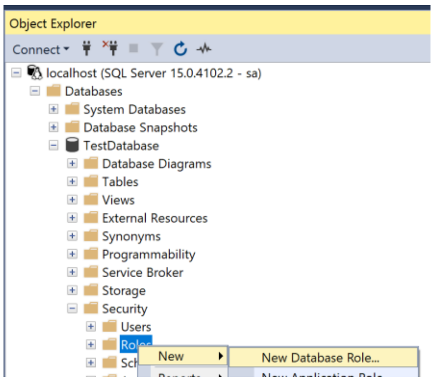
On the folder `Databases/<DatabaseName>/Security/Users` right click and select New User.



On the setup dialogue enter the name and the login we want to associate the user with. You should now be able to log on to the server with the login and then access the database.

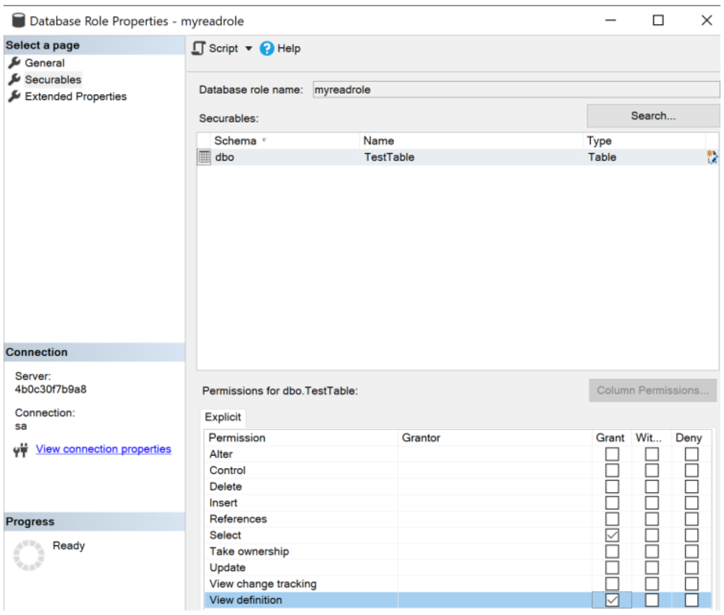
Create Role

To be able to do anything we need to create a role that can read our table. On the folder `Databases/<Database>/Security/Roles` select New Database Role.



Add permissions to role

Right click on role and select properties and then on the dialogue select Securables.



Database role name: myreadrole

Securables:

Schema	Name	Type
dbo	TestTable	Table

Server: 4b0c30f7b9a8

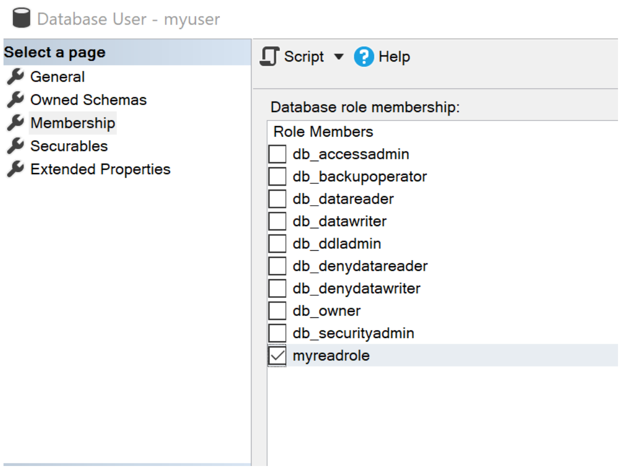
Connection: sa

Permissions for dbo.TestTable:

Permission	Grantor	Grant	Wit...	Deny
Alter		☐	☐	☐
Control		☐	☐	☐
Delete		☐	☐	☐
Insert		☐	☐	☐
References		☐	☐	☐
Select		✓	☐	☐
Take ownership		☐	☐	☐
Update		☐	☐	☐
View change tracking		☐	☐	☐
View definition		✓	☐	☐

Associate role with user

Right click on user and select properties then on the Membership page add the new role we created.


Database User - myuser

Select a page
- General
- Owned Schemas
- Membership
- Securables
- Extended Properties

Script ▾ Help

Database role membership:

Role Members
☐ db_accessadmin
☐ db_backupoperator
☐ db_datareader
☐ db_datawriter
☐ db_ddladmin
☐ db_denydatareader
☐ db_denydatawriter
☐ db_owner
☐ db_securityadmin
✓ myreadrole

We can now read the table.

.NET Core

We can connect to our automatic Local DB from code as follows.

```
SqlConnection connection = new SqlConnection(@"Data
Source=(localdb)\MSSQLLocalDB;Database=JsonObjects;");
connection.Open();
```

We can insert into our table as follows

```
var json = @"{""Hello"":""World2""}";
var sql = $"insert into MYDOCS (DOC) Values('{json}')";
var command = new SqlCommand(sql, connection);
command.ExecuteNonQuery();
```

Questions

Select

Phases

List the phases of a select query.

SELECT

FROM

WHERE

GROUP BY

HAVING

ORDER BY

What is the logical order of the phases and what does each do?

1. *FROM select the table we want to query*
2. *WHERE filter the rows returned.*
3. *GROUP BY produce group for each combination.*
4. *HAVING filter the groups.*
5. *SELECT Specify the columns for result*
6. *ORDER BY sort results for presentation*

What is the key benefit of the where clause?

Enable indices to improve performance and reduce network traffic.

Given the following table write a query that first filters out everyone over 40. Then group by age and countryId and show the count for each group. Only include groups with countryId of 1 or 10

firstname	secondname	ageΞΞ	countryId
John	Smith	25	1
Dave	Jones	25	1
Aaro	Litmanen	30	10
Kimi	Raikonen	30	10
Han	Solo	40	2
Luke	Skywalker	35	2
Indiana	Jones	50	3
Carl	Jones	50	3

```
SELECT age, countryId, COUNT(*)

FROM Person

WHERE age < 50

GROUP BY age, countryId

HAVING countryId IN (1,10)

ORDER BY age, countryId
```

Basic Select

List the firstname of all rows but rename it “First Name”

```
SELECT firstname AS 'First Name'

FROM Person
```