Temel SQL Sorguları Üçüncü Kısım

Tarih formatı

Karakterler yanyana olarak ifade edilirse numerik bir çıktı elde edilir (MM,YYYY).

Kelime olarak ifade edilir ise çıktı tarihin kendisi olarak verilir (Month, Day, Year).

Apexde ise format, iki hane ay, iki hane gün ve dört hane de yıl şeklindedir.
(MM/DD/YYYY)

```
SELECT TO_DATE('01-APR-2022', 'DD-MON-YYYY')

FROM DUAL;
```

NOT: Virgülden sonraki kısımda inputun/girdinin formatını belirlemiş olduk.

```
SELECT  TO_CHAR(TO_DATE('01-APR-2022','DD-MON-YYYY'),'DD-MON-YYYY')

FROM DUAL;
```

NOT: Outputun/çıktının formatını düzenledik.

```
SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date, 6),'FRIDAY'),

'fmDay, Month ddth, YYYY') AS "Next Evaluation"

FROM employees;

WHERE employee_id=100;
```

NOT: Şimdiki tarihe 6 ay ekler, bu tarihten itibaren de ilk Cuma gününü bulur. Bu tarihi de karaktere çevirip günü gün olarak verir.

```
SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date, 6),'FRIDAY'),

'DD-MONTH_YYYY')

FROM employees;

WHERE employee_id=100;
```

NULL FUNCTİONS

1) NVL

2) NVL2

3) NULLIF

4) COALESCE ()

1) NVL

Seçtiğimiz kısım null değere sahip ise ona null getirmek yerine vereceğim değeri yazar.

```
SELECT last_name, NVL(commission_pct, 0)
FROM employees
WHERE department_id IN(80,90);
```

```
SELECT NVL(date_of_independence, 'No Date')
FROM wf_countries;
```

```
SELECT last_name, NVL(commission_pct, 0) *250 as "Commission"
FROM employees
WHERE department_id IN(80,90);
```

2) NVL2

Seçtiğim kısımdaki ilk değer null değilse ikinci değeri devreye sokar

NVL: “Verdiğim sütundakı değer nullsa şunu yap.”

NVL2: “Verdiğim sütundakı değer null değilseü çüncü satırı, değilse diğerini yap.”

```
SELECT last_name, salary,
NVL2(commission_pct, salary +(salary*commission_pct),salary) as "İNCOME"
FROM employees
WHERE department_id IN(80,90);
```

NOT: eğer commission_pct null ise yerine salary yazar, değilse hesaplamayı devreye sokar.

3) NULLIF

Length eşitse null yapar, eşit değilse birincinin değerini basar.

```
SELECT first_name, LENGTH(first_name) as length fn, last_name
LENGTH(last_name) as lenght ln, NULLIF(LENGTH(first_name))
FROM employees
WHERE department_id IN(80,90);
```

4) COALESCE

Null olmayan değer bulana kadar kovalar.

```
SELECT last_name, commission_pct, salary
COALESCE(commission_pct, salary, 10)
FROM employees;
```

NOT: commission_pct null değere sahipse salary'yi yazar, o da null değer ise 10 yazar.

NOT: comission_pct null değil direkt yazdı, null olanlarda salary değerini yazdı. 10 yazması için salarynin de null olması gerekirdi.

CONDİTİONAL EXPRESSİONS

DECODE CASE, if else gibi bir karar mekanizmasıdır.

1) **CASE**

CASE expr WHEN comparison_expr1 THEN return_expr1

NOT: expr değişkeni ilkine eşitse ikincsini retrun eder.

SELECT last_name, department_id as "DepNo"

CASE department_id

WHEN 90 THEN 'Management'

WHEN 80 THEN 'Sales'

WHEN 60 THEN 'IT'

ELSE 'Other dept'

END AS "Department"

FROM employees;

2) **DECODE**

Sıralı bir kontrol yapısıdır.

SELECT last_name, department_id as "DepNo"

DECODE (department_id, 90,'Management', 80, 'Sales', 60, 'IT', 'Other dept') AS "Department"

FROM employees;

JOİN

SQL'ın temel aleti JOIN’dir. Dağıttıklarımızı doğru bir şekilde birleştirebilmek için gereklidir.

1) NATURAL JOİN

Kayıpsız bir normalizasyon kontrolü için kullanırız. Aynı adı alan tüm sütunları karşılaştırır. Aynı sütunları bulup içindeki değerleri eşlestirip birleştirir.

a) SELECT first_name, last_name, job_id

FROM employees;

b) SELECT job_id, job_title FROM jobs;

c) SELECT first_name, last_name, job_id, job_title

FROM employees NATURAL JOIN jobs

WHERE department_id>80;

NOTLAR:

// Burada job_ title’ı join ediyorum.

//Bu işlemi sütunları ve içerisindeki değerleri karşılaştırarak uyguluyor.

//desc employees;

//desc jobs;

//ortak sütun job_id

SELECT department_name, city

FROM departments NATURAL JOIN locations;

NOTLAR:

//department ve locations arasında bir join yaptık.

//location ıd uzerınden var olan ortak sütun üzerinden ilişki kuruldu.

//desc departments

//desc locations

2) **CROSS JOIN**

SELECT last_name, department_name

FROM employees CROSS JOIN departments;

SELECT last_name, department_name

FROM employees departments;

NOT: Burada cross joın otomatik yaptı. WHERE veya JOIN ON, JOIN USING, NATURAL JOIN kullanırsam yapmazdı.

JOIN CLAUSES

1) **USING**

Natural Joın aksine burada spesifik olarak sadece belirli bir satırın değerlerini birleştirebiliyoruz. Kısaca sütun seçebiliyoruz.

SELECT first_name, last_name, department_id, department_name

FROM employees JOIN departments USING (department_id);

NOT: 20 row employees da vardı. Bir tanesi null değer olduğu için sonuç 19 çıktı.

NOT: Eğer burada NATURAL JOIN deseydik;

SELECT first_name, last_name, department_id, department_name

FROM employees JOIN departments;

//Yıne 19 urettı. Ortak sutun sadece 1 tane cunku.

//İkisinde de olmayan şeyi usıng dersem hata verir. Usıngde kullancaĞım sütun iki kısımda da ortak olarak olması lazım.

SELECT first_name, last_name, department_id, department_name

FROM employees JOIN departments USING (employee_id);

NOT: USING içinde Alies almaması lazım. Hata verir.

ALİAS ile:

SELECT last_name, e.job_id, job_title

FROM employees e, jobs j

WHERE e.job_id = j.job_id;

NOT: WHERE ile job id'ler eşitse bunu yap dedik ve joın yapmış olduk, Cross joından kurtulduk. Where olmasa 140 satır gelirdi.

SELECT last_name, job_id, job_title

FROM employees e JOIN jobs j USING(job_id)

NOT: e.job_id şeklinde yazarsak hata verir.

NATURAL JOIN VE USING ARASINDAKİ FARK

desc job_history; (employee job ve department ıd si var. Bunlar employee de de avrdı)

SELECT first_name, last_name (sadece employeesda var) , start_date, end_date (historyden gelir)

FROM employees NATURAL JOIN job_history

1.) SELECT first_name, last_name, start_date, end_date

FROM employees JOIN job_history USING (job_id)

2.)SELECT first_name, last_name, start_date, end_date

FROM employees JOIN job_history USING (job_id, employee_id)

3.)SELECT first_name, last_name, start_date, end_date

FROM employees JOIN job_history USING (job_id, employee_id, department_id) (ortakları ekledik)

2) **ON**

Burada sütunun aynı olup olmaması artık önemlı değil. Yani farklı sutünlarda JOIN ON yapabilirim.

APEX ile ilgili;

desc employees; (manager_id var)

desc departments; (yine manager_id vardı eskiden yanlış bir durumdu, joın ornegı gosterebılmek ıcın. Deparmenttaki manager_id, dept_manager_id olarak değiştirildi. İçindeki valuelar eşit)

SELECT employee_id, manager_id, dept_manager_id

FROM employees JOIN departments ON(manager_id=dept_manager_id)

NOT: ON ile sütunu ben söyledim, içindeki değerleri eşleşiyorsa bu sütunu birleştir. Sütun adları farklı ama içindeki değerler aynı.

SELECT employee_id, manager_id, dept_manager_id

FROM employees NATURAL JOIN departments;

NOT: Natural joın ile ne yapardı?

SELECT last_name, job_title

FROM employees e JOIN jobs j ON(e.job_id=j.job_id)

NOT: Birinin job id'si ile diğerininkini eşleştirdi

SELECT last_name, job_title

FROM employees e JOIN jobs j USING(job_id)

NOT: USING ile ortak olanla da getirebildim.

select * from job_grades; (Çalışanların grade'lerini nasıl tespit edeceğim? Aralarında sütun içindeki değerleri aynı olan yok.)

SELECT last_name, salary, grade_level, lowest_sal, highest_sal

FROM employees JOIN job_grades ON(salary=lowest_sal);

NOT: Bu gradeler arasında en düşük kim alıyor?

BİRDEN FAZLA TABLONUN JOIN EDİLMESİ

SELECT last_name department_name as department, city

FROM employees JOIN departments USING(department_id);

USING, NATURAL JOIN ve ON ARASINDAKİ FARK

NATURAL ve USING'de aynı sütun değerleri karşılaştırılırken ON'da ise farklı sütunlar içindeki değerler karşılaştırılıyor.

3) INNER & OUTHER JOIN

INNER JOIN default değerdir. Belirtmeye gerek yok.

SELECT e.last_name, d.ddepartment_id, d.department_name

FROM employees e JOIN departmens d ON (e.department_id=d.department_id);

NOT: 19 kayıt döner çünkü birinin departmanı yok.

a)emp de var dept te olmayanlarda dahil

SELECT e.last_name, d.ddepartment_id, d.department_name

FROM employees e

LEFT OUTHER JOIN departmens d ON (e.department_id=d.department_id);

NOTLAR:

//left outher dedım ve 20 kayıt döndü. Department id si yok ama yine de geldi. Çünkü OUTHER hepsini getirsin, LEFT OUTHER ise eşleşmeyeni de getir demek.

//FROM dan sonra employees gelmiş 1. ve department 2. Left olan employees, left outher ile employeesda olup departmensda olmayanı getır dedım.

a) emp de var dept te olmayanlarda dahil

SELECT e.last_name, d.ddepartment_id, d.department_name

FROM departments e

RIGHT OUTHER JOIN employees d ON (e.department_id=d.department_id);

//üstteki ile aynı sonucu verir.

//Neden Ailes veririz? Department ıd lerı karsılstırablmek ıcın. Department_id ikisindede olduğu için e ve d olarak ayırmam ve onların esıtlıgını boyle kontrol edebılmem lazım

//Left dedıgı zaman employeesa baktı, employeesda olanlar deparetmanta olmasa bile getir dedim.

b) dept de var empt de olmayanlar da dahil

SELECT e.last_name, d.ddepartment_id, d.department_name

FROM departments e

LEFT OUTHER JOIN employees d ON (e.department_id=d.department_id);

b) dept de var empt de olmayanlar da dahil

SELECT e.last_name, d.ddepartment_id, d.department_name

FROM employees e

RİGHT OUTHER JOIN departmens d ON (e.department_id=d.department_id);

---FULL OUTHER---

Hem sağ hem sol söz konusu

c) Her iki tabloda da fazla olanlar gelir:

SELECT e.last_name, d.ddepartment_id, d.department_name

FROM employees e

FULL OUTHER JOIN departmens d ON (e.department_id=d.department_id);

Senario

Çalışanların eski işlerini göster?

SELECT last_name, e.job_id as "JOB", jb.job_id as "OLD JOB", end_date

FROM employees e LEFT OUTHER JOIN job_history jh

ON (e.employee_i = jh.employee_id);

NOTLAR:

//King iş değiştirmemiş.

//Kochar iş değiştirmiş.

4) SELF JOINS

Bir kayıtta bulunmayan bir değere daha ulaşmam gerekirse SELF JOIN kullanırız.

Ahmet müdür 101

Ali çalışan 202 müdürünün kodu 101 ---> employees

202 id li çalışanı getir, yanında müdürünün adını da getir. Bunun için self-joın gerekir.

Çalışan kodu 101 olan kim? Önce bunun cevabını bulmam gerekir.

SELECT worker.last_name || 'works for' || manager.last_name as "WORKS FOR"

FROM employees worker JOIN employees manager (2 tane ayrı tablo olusturduk zanneder sql)

ON (worker.manager_id = manager.employee_id);

SELECT worker.last_name || 'works for' || manager_id as "WORKS FOR"

FROM employees

NOT: manager id olsaydı eşleştirmeye ve joın'e ihtiyaç kalmazdı.

```
SELECT worker.lastname, worker.manager_id, manager.last_name as "manager name"

FROM employees worker JOIN employees manager

ON ()
```

NOT: ON kısmında eşleştirmeyı doğru yapmak asıl meseledir. En çok hata bu kısımda meydana gelir.

----------VİZE HAKKINDA----------

+7.section oracle has olan kısımlar sorulmayacak.

--7.2 outer joın--

+Joın Comprasiondaki terimler ne demek hakim ol sınavda sorarmış.

Self joın hazırlıklı olmanızı tavsiye ediyorum.

+Group functionslar vizeden sonra ağırlıklı olur. 3 kuralın üstünde durdu hoca not alın dedi.

Büyük ihtimal teorik sorabilir bu konuda.

+Vocabulary çalışmaları sınavda çıkacak. (Bir açıklama, boşluk olacak.)

Cevap sütunu koyacak hoca yine.

+iki output verir hangisi joın on hangisi natural joın e ait?

+aralarındaki fark nereden neden kaynaklanıyor?

+.. yı IN kullanmadan nasıl yapabilirdik?

+rollup, cube, groupıng sets ve hiearsık query vizede yok.

+SELF JOIN cıkabilir

+Veri tabanı tasarımına yönelik temel sorular çıkacak

Pratiklerdeki vocabulary kısımları gibi çıkacak.

+Cevaplar tek cevaba dayanacak.

+Tanım veya açıklama verilir.

+SQL deki keywordler neler

+Functionlar, textlerle işlem yaparken kullandığımız functionlar;

"bir metinden beklenen uzunlukta bir stringi çeker. Bunun için kullnman gereken charahcter function nedir?"

+Doğru yanlış soruları.

+Veri tabanına ilgili kısımlar dönem başımdakiler yüksek puanlı olacak.

+Doğrudan doğruya sql cumlesi yazman bir iki tane istencek onlarda cok basıt temel olacak.

+Employees tablosunda sadece şu kayıtlar olduğunu kabul edelim denilip sorulacak.

O veriler üzerinden bir sql cumlesı verılıp cıktıyı resmetmen istenecek.

+SQL in temel kurallarından soru gelecek.

+JOIN lere ağırlık ver. Aralarındaki farkları iyi bil.

Bu tablolar arasında su joınler var aralarında su farklar var sebebleri neler.

"Şu joında sunlar yapılır, bu joında bunlar yapılır...."

+Eqiual joın non-eqiual joın

+Karsılastırma operatorleri

+groupby cıkabılır kesın olmamakla beraber.

+Group functıonslardan hangıleri verı tpıne baglı olmadan calısır hem alfatbetik hem numerik...(MIN,MAX)

----------JOIN COMPARİSON----------

Cartesian Product

1) CROSS JOIN

Equjoin (Ortak sütunların eşleştirilmesi söz konusu):

1) NATURAL JOIN: Aynı sütunların taşıdığı değer bakımından eşleştirme aranır

2) JOIN USING: Eşleştirilecek sütunlar belirtilerek taşıdıkları değerler eşleştirilir.

3) JOIN ON: Değerler karşılaştırılır ama ortak sütun gerekmez, sütun adları farklı olabilir.

Non-equjoin (Farklı sütunların değerlerinin eşleştirilmesi çalışılır):

1) ON:

2) BETWEEN:

----------GROUP FUNCTİONS----------

3 KURAL:

*Group functions ignore null values

*Group functions cannot be used in the WHERE clause

*MIN, MAX and COUNT can be used with any data type;

SUM, AVG, STDDEV and VARIANCE can be used only with numeric data types.

Multi-row functions, input birden fazla ama output bir adet.

SUM: Toplam alınır. (İnput sayısı ne olursa olsun bir adet output yani toplam çıkacaktır)

AVG: Ortalama alınır.

COUNT

MIN

MAX

STDDEV: Standart sapma.

VARİANCE: Varyans.

SELECT MAX(salary) FROM employees;

SELECT MIN(hire_date) FROM employees;

SELECT SUM(salary) FROM employees;

-GROUP Function'lar WHERE ile kullanılamaz :

SELECT last_name, last_name

FROM employees

WHERE salary = MIN(salary);

SELECT ROUND(AVG(salary),2)

FROM employees

WHERE department_id=90;

(GROUP function'ı select de kullandı where de kullanmadı)

NULL:

SELECT AVG(commission_pct)

FROM employees;

(20 ye değil 4 e böldü. NULL değerleri görmezden geldi)

SELECT MAX(salary), MIN(salary), MIN(hire_date)

FROM employees;

WHERE department_id= :dept;

----------COUNT----------

SELECT COUNT(job_id) FROM employees;

(rowları saydı)

SELECT COUNT(department_id) FROM employees;

(Nulları saymadı. Group functions ignore null values)

SELECT COUNT(*) FROM employees;

(Null a bakmaz hepsini getirir bu daha doğrudur emin olmak adına)

SELECT COUNT(*) FROM employees

WHERE hire_date < '01-01-1996'

SELECT COUNT(*) FROM employees

WHERE hire_date < '01-Jan-1996' //Hata verir numerik tarih vermediği için. Date formatı sıkıntısı. MON/DD/YYYY DD sayısal bekliodum Jan verdin

----------DISTINCT----------

DISTINCT Sadece farklı olanları getirirdi.

SELECT job_id FROM employees;

SELECT DISTINCT job_id FROM employees;

SELECT SUM(salary)

FROM employees

WHERE department_id=90;

SELECT SUM(DISTINCT salary)

FROM employees

WHERE department_id=90; // sadece farklı salaryleri topladı

SELECT COUNT(DISTINCT job_id)

FROM employees;

SELECT DISTINCT job_id

FROM employees;

----------NVL----------

SELECT AVG(comission_pct) FROM employees; //4 E BÖLER

SELECT AVG(NVL(comission_pct, 0)) FROM employees; //20 YE BÖLER

Database Design and Programming with SQL – Course Description

Overview

This course engages students to analyze complex business scenarios and create a data model—a conceptual representation of an organization's information. Participants implement their database design by creating a physical database using SQL. Basic SQL syntax and the rules for constructing valid SQL statements are reviewed. This course culminates with a project that challenges students to design, implement, and demonstrate a database solution for a business or organization.

Available Curriculum Languages:

- English, Simplified Chinese, Brazilian Portuguese, Spanish

Duration

- Recommended total course time: 180 hours*
- Professional education credit hours for educators who complete Oracle Academy training: 60

** Course time includes instruction, self-study/homework, practices, projects, and assessment*

Target Audiences

Educators

- College/university faculty who teach computer programming, information communications technology (ICT), or a related subject
- Secondary school teachers who teach computer programming, ICT, or a related subject

Students

- Students who wish to learn the techniques and tools to design, guild and extract information from a database
- Students who possess basic mathematical, logical, and analytical problem-solving skills
- Novice programmers, as well as those at advanced levels, to learning the SQL Programming language to an advanced level

Prerequisites

Required

- Ease with using a computer
- General knowledge of databases and query activity

Suggested

- None

Suggested Next Courses

- Database Programming with PL/SQL

Lesson-by-Lesson Topics

Database Design

Introduction

- Introduction to the Oracle Academy
- Data vs. Information
- History of the Database
- Major Transformations in Computing

Entities and Attributes

- Conceptual and Physical Models
- Entities, Instances, Attributes, and Identifiers
- Entity Relationship Modeling and ERDs

Relationship Basics

- Identifying Relationships
- ER Diagramming Conventions
- Speaking ERDish & Drawing Relationships
- Matrix Diagrams

Super/Sub Types and Business Rules

- Supertypes and Subtypes
- Documenting Business Rules

Relationship Fundamentals

- Relationship Transferability
- Relationship Types
- Resolving Many-to-Many Relationships
- Understanding CRUD Requirements

UIDs and Normalization

- Artificial, Composite, and Secondary UIDs
- Normalization and First Normal Form
- Second Normal Form
- Third Normal Form

Arcs, Hierarchies, and Recursive Modeling

- Arcs
- Hierarchies and Recursive Relationships

Changes and Historical Modeling

- Modeling Historical Data
- Modeling Change: Time
- Modeling Change: Price
- Drawing Conventions for Readability

Mapping

- Introduction to Relational Database Concepts
- Basic Mapping: The Transformation Process
- Relationship Mapping
- Subtype Mapping

Creating Database Projects

- System Development Life Cycle
- Project Overview and Getting Started

- Presentation Project Management
- Final Presentation Components

Presenting Database Projects

- Creating Tables for the Final Presentation
- Preparing Written Documentation
- Preparing Visual Materials
- Final Presentations

Database Programming with SQL

Introduction

- Oracle Application Express
- Relational Database Technology
- Anatomy of a SQL Statement

SELECT and WHERE

- Columns, Characters, and Rows
- Limit Rows Selected
- Comparison Operators

WHERE, ORDER BY, and Intro to Functions

- Logical Comparisons and Precedence Rules
- Sorting Rows
- Introduction to Functions

Single Row Functions Part I

- Case and Character Manipulation
- Number Functions
- Date Functions

Single Row Functions Part II

- Conversion Functions
- NULL Functions
- Conditional Expressions

JOINs

- Cross Joins and Natural Joins
- Join Clauses
- Inner versus Outer Joins
- Self-Joins and Hierarchical Queries
- Oracle Equijoin and Cartesian Product
- Oracle Nonequijoins and Outer Joins

Group Functions

- Group Functions
- Oracle Nonequijoins and Outer Joins
- Using Group By and Having Clauses
- Using Rollup and Cube Operations, and Grouping Sets
- Using Set Operators

Subqueries

- Fundamentals of Subqueries
- Single-Row Subqueries
- Multiple-Row Subqueries
- Correlated Subqueries

Ensuring Quality Queries Part I
- Ensuring Quality Query Results

DML
- INSERT Statements
- Updating Column Values and Deleting Rows
- DEFAULT Values, MERGE, and Multi-Table Inserts

DDL
- Creating Tables
- Using Data Types
- Modifying a Table

Constraints
- Intro to Constraints; NOT NULL and UNIQUE Constraints
- PRIMARY KEY, FOREIGN KEY, and CHECK Constraints
- Managing Constraints

Views
- Creating Views
- DML Operations and Views
- Managing Views

Sequences and Synonyms
- Working With Sequences
- Indexes and Synonyms

Privileges and Regular Expressions
- Controlling User Access
- Creating and Revoking Object Privileges
- Regular Expressions

TCL
- Database Transactions

Final Project and Exam Review
- Testing
- Final Project Database Creation
- Final Exam Review

Ensuring Quality Queries Part II
- Ensuring Quality Query Results - Advanced Techniques

To search and register for events scheduled in your area, visit the Academy events calendar.

Normalizasyon ve Bazı Temel SQL Bilgileri

Normalizasyon nedir?

Normalizasyon, veri tabanlarındaki tabloların içeriğini organize etme tekniğidir. Başarılı veri tabanı tasarımının bir parçasıdır. Normalizasyon olmadan, veri tabanı sistemleri; yanlış, yavaş ve verimsiz olabilir ve beklediğiniz verileri üretmeyebilir.

Bir veri tabanının, veri tekrarını en aza indirgemek ve her tabloda yalnızca ilgili verilerin depolandığından emin olmak için normalleştirilmesi önemlidir. Bir tablo belirli bir konu hakkında olmalı ve sadece destekleyici konular içermelidir.

Bir tabloyu bir amaç ile sınırlandırarak, veri tabanınızdaki yinelenen veri sayısını azaltırsınız. Bu, veri tabanı modifikasyonlarından kaynaklanan bazı sorunları ortadan kaldırır. Bu hedeflere ulaşmak için bazı yerleşik kurallar kullanılır. Bu kuralları uygularken yeni tablolar oluşturulur.

Neden bir veri tabanı normalleşme ihtiyacı duyar?

Bir veri tabanını normalleştirmenin üç ana nedeni vardır. Birincisi yinelenen verileri en aza indirmek, ikincisi veri modifikasyon sorunlarını en aza indirmek veya önlemek ve üçüncüsü sorguları basitleştirmek.

Veri tabanı mantığının en önemli ilkelerinden biri;

“Her nesne tipi için ayrı bir tablo yaratmalısın.”

Normalizasyon Formları

1.Normal Form:

“Depend on the key ”

Primary key’ler belirlenmeli ve tekrarlı veriler ortadan kaldırılmalı.

2.Normal Form:

“The whole key ”

Bir primary key’e bütün diğer ilgililerde bağlı olmalı ve herhangi alt küme durumu söz konusu olmamalı. Kısmi bağımlılıklar ortadan kaldırılmış durumda olmalı.

3.Normal Form:

“Nothing but the key”

Geçişli bağımlılıklar ortadan kaldırılmış durumda olmalı.

1NF

Students		
FName	**SName**	**Class**
Timothy	Smith	Computer Science
Jessica	Green	Computer Science
Jessica	Green	Maths
Mark	Lynch	Maths

2NF

Students			
ID	**FName**	**SName**	**Class**
1	Timothy	Smith	1
2	Jessica	Green	1
3	Jessica	Green	2
4	Mark	Lynch	2

Class	
ID	**Class**
1	Computer Science
2	Maths

3NF

Students		
ID	**FName**	**SName**
1	Timothy	Smith
2	Jessica	Green
3	Mark	Lynch

Subject	
ID	**Class**
1	Computer Science
2	Maths

Class		
ID	**Student**	**Class**
1	1	1
2	2	1
3	2	2
4	3	2

Veri Tabanı ile alakalı bazı terimler

Table: Verileri koyduğumuz adresler.

Column: Tablonun sütunları.

Row: Tablonun satırları.

Field: Sütunun başka bir adı, daha geniş bir anlam ifade eder.

Primary key: Sütunların bağımlılığı hakkında, belirlenen ve belirsiz satırlar vardır.

Foreign key: Tablolar arasındaki ilişkilerde kullanılır.

Tablo Özellikleri

1) Sütunlar tek değerli olmalı. Herhangi sütundaki sütunun içerdiği değer tek olacaktır.

2) Sütunlardaki değerler aynı cinsten değerler olacak.

3) Her bir sütunun aslında kendisi başlı başına bir oluşum ifade ettiğinden, tekrar etmemeli. ("Each row is unique")

4) Sütunların sırasının önemi yok.

5) "Sequence of rows is insignificant": Diskteki yerleşimin önemsizliğinden bahsediyor

6) Sütunlara ad verirken, bir tabloda sadece bir tane özgü başlık adı olabilir.

NOT: Datalar veri tabanında blok blok oluşur. Bir bloğa kaç sütun tekabül edeceği orada hesaba katılır.

Programlama Dillerinin Gruplandırılması

Data Manipulation Language (DML): Uç kullanıcıların yaptığı işlemlerdir. İNSERT, UPDATE, DELETE veya MERGE komutları. Oluşan objeler üstünde çalışma yapar.

Data Definition Language (DDL): Developer tarafından, temel objelerin oluşturulmasıyla ilgili işlemler.

Transaction Control Language (TCL): COMMIT, ROLLBACK, SAVEPOINT. Veri tablodaki değerleri kontrol eder, dataları bu komutlarla güncelleyebiliriz. Örneğin X saat sonrası yaptığın tüm değişiklikleri SAVEPOINT ile geri alabiliriz.

Data Control Language (DCL): Yetkilendirme ile alakalı. Veri tabanında bir SELECT yapmamız bile GRANT ile alakalı. O kullanıcının SELECT yetkisi var mı? Varsa hangi alanlarda? REVOKE, yetkiyi geri alır.

Projection: Dikey seçim, sütun seçimi

Selection: Yatay seçim, satır seçimi.

Yaygın Kullanım Şekli: SELECT <Projection elemanı olan sütun adı>;

Aliases: "AS ……." şeklinde bir işlem sonrası o sütuna verdiğimiz yeni sütun başlığımızdır.

NOT: NULL değerleri çıktıda "-" olarak gözükür. NVL ile değiştirilebilir.

NOT: Her SQL cümlesi tek bir cümledir.

Bu Kısımda Değerlendirdiğimiz Konu Başlıkları

1-ROW

2-PRİMARY KEY

3-TABLE

4-FOREİGN KEY

5-İLİŞKİSEL VERİ TABANI

6-FİELD

7-DML

8-DDL

9-TCL

10-DCL

11-NORMALİZASYON

12-NORMAL FORMLAR

TEMEL SQL SORGULARI BİRİNCİ KISIM

SQL KOMUTLARININ İCRA SIRASI

Keywordslerin yazılış sırasından farklı olarak SQL'in syntaxtaki sıralaması şu şekilde olmalıdır:

"SELECT clause FROM clause"

SELECT, FROM ve * Kullanımı

a) Employee tablosuna ait yalnızca belirttiğim sütunları çağırıyorum:

```
SELECT employee_id, first_name, last_name
FROM employees;
```

b) Employee tablosuna ait tüm sütunları çağırıyorum.

```
SELECT * FROM employees
```

WHERE VE Operatörlerin Kullanımı

a) Belirttiğim sütunları yalnızca employee_id'si 101 olanlardan getirecek.

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE employee_id=101;
```

b) Belirttiğim sütunları yalnızca employee_id'si 1'den büyük veya eşit olanlardan getirecek.

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE employee_id >= 1;
```

c) Burada Taylor kısmında büyük küçük harf uyumu dikkate alınmalı. Çünkü burada belirteceğimiz bilgi doğrudan veri tabanından alınıp getirilir. Bu harf duyarlılıklarını, LOWER ve INITCAP ile ortadan kaldırabiliriz.

```
SELECT first_name, last_name
FROM employees
WHERE last_name='Taylor';
```

KARŞILAŞTIRMA OPERATÖRLERİ

-BETWEEN

-IN

-LIKE

-NULL

-BETWEEN...AND

a) BETWEEN kullanımı:

```
SELECT first_name, salary
FROM employees
WHERE salary BETWEEN 9000 AND 11000;
```

```
SELECT first_name, salary
FROM employees
WHERE salary >= 9000 AND salary <= 11000;
```

NOT: Bu iki cümle de aynı çıktıyı verir.

b) LIKE kullanımı

Veritabanındaki kayıtlı bilgiyi çekip kullanabilmek için verilerin hepsini ezbere bilmemize gerek yok. Birebir eşleşeni bulmak zor olabilir. LIKE ile benzerlikleri sorgulayarak veriye daha kolay ulaşabiliriz. (% ve _)

```
SELECT last_name
FROM employees
WHERE last_name LIKE '_o%'
```

NOT: Verimde o harfi olsun ama 2.harf olsun. %'den sonra kaç harf olduğu beni ilgilendirmez.

```
SELECT last_name FROM employees
WHERE last_name LIKE '_O%'
```

NOT: Büyük küçük harf uyumu burada da söz konusu.

```
SELECT last_name
FROM employees
WHERE last_name LIKE '%A%'
```

Peki ya % veya _ işaretleri benim asıl verimin içinde olursa ne yapacağım?

```
SELECT last_name
FROM employees
WHERE last_name LIKE '%\_R%'  ESCAPE '\'
```

NOT: Bu cümle ile ORACLE EX'e bu cümleyi kurmuş oluyorum: "R ve öncesinde _ olanın veri olduğunu anla ve bu işareti harf sayısı belirtmişim gibi algılama."

NULL

Her erişmeye çalıştığımız veri bir değere karşılık gelmeyebilir. Bu durumlarda o veri NULL değere sahip olmuş olur ve veri tabanı tablosunda değerine karşılık "-" işareti ile belirtilir. Dilersek bu "-" işaretinin yerine kendi belirttiğimiz bir cümleyi veya değeri NVL komutu yazabiliriz. Daha sonraları bu konuya değineceğim.

Eşittir yerine "IS NULL" kullanırız.

Eşit değil yerine "IS NOT NULL" kullanırız.

```
SELECT last_name, manager_id
FROM employees
WHERE manager_id IS NULL;
```

NOT: manager_id değeri NULL olanların last_name ve manager_id'sini getirir.

```
SELECT last_name, commission_pct
FROM employees
WHERE commission_pct IS NOT NULL;
```

NOT: commission_pct değeri NULL olmayanların last_name ve commission_pct'sini getirir.

MANTIKSAL KARŞILAŞTIRMALAR VE ÖNCELİK KURALLARI

-NOT

-IN

-NOT IN

-AND

-OR

a) OR kullanımı

```
SELECT city, state_province, country_id
FROM locations
WHERE country_id IN('UK','CA');
```

```
WHERE country_id = 'UK' OR country_id = 'CA';
```

NOT: Bu iki cümle de aynı çıktıyı verir.

```
SELECT first_name, department_id, salary
FROM employees
WHERE department_id> 50 AND salary > 12000;
```

```
SELECT first_name, department_id, salary
FROM employees
WHERE department_id > 50 OR salary > 12000;
```

NOT: OR komutunda tek tarafı doğrulamak yeterli olacaktır.

```
SELECT department_name, dept_manager_id, location_id
FROM departments
WHERE location_id=2500 OR dept_manager-id=124;
```

b) NOT VE IN kullanımı

SELECT department_name, location_id

FROM departments

WHERE location_id NOT IN (1700,1800);

NOT: location_id'leri 1700 ve 1800 olmayanların belirttiğim satırlarını getirir.

SELECT department_name, location_id

FROM departments

WHERE location_id IN (1700,1800);

NOT: location_id'leri 1700 ve 1800 olanların belirttiğim satırlarını getirir.

c) AND kullanımı

SELECT last_name, department_id, first_name, last_name FROM employees

WHERE department_id IN(50,80) AND first_name LIKE 'C%'

OR last_name LIKE '%m%';

NOT : AND kısmında iki tarafı da birlikte okur, eğer koşul sağlanmıyorsa OR kısmı çalışır.

ORDER BY

Verileri veri tabanımızdan çekip çağırırken SQL 'e bu işlemi belirli bir kısıta göre sıralama yaparak getirmesini ORDER BY komutu ile söyleyebiliriz.

Çıktı üzerinde çalıştığı için ORDER BY cümlenin sonuna yazılır.

Aynı zamanda SELECT edilmemiş sütunlara da ORDER_BY komutunu verebiliriz.

SELECT last_name, hire_date

FROM employees

ORDER BY hire_date;

NOT: Burada kısıtı hire_date olarak belirledik ve satırdaki verileri hire_date'e göre sıralama yaparak çağırdı

Aynı şekilde iç içe bir sıralama yapmak da mümkün;

```
SELECT last_name, hire_date
FROM employees
ORDER BY hire_date, last_name;
```

NOT: Önce hire_date göre, onun içinde de last_name e göre sıralama yapar.

ORDEY BY kullanırken varsayılanın dışında ters sıralama yapmamız da DESC komutu ile mümkün;

```
SELECT last_name, hire_date AS “işegiriş”
FROM employees
ORDER BY işegiriş desc, last_name;
```

NOT:

ASCENDİNG: Küçükten büyüge, default.

DESCENDİNG: Büyükten küçüğe, ayrıca belirtilmeli.

İNTRODUCTİON of FUNCTİONS

Sinlge-Row Functionlar: Girdisi bir tane ise çıktısı da bir tane olması gerekir. Yani her sütun için yalnızca bir çıktı elde ederiz. (Money --> Drink Machine --> Drink)

Multi-Row Functionlar: Girdi birden fazladır ama çıktı yalnızca bir tanedir.

CONCATENATİON

Sütunları birleştirme işlemlerini bu komut ile gerçekleştiririz.

string1 || string2 || string_n

```
SELECT department_id ||' '|| department-name as DEPARTMENT
FROM departments;
```

```
SELECT last_name || ' Has a monthly salary of '|| salary*2 || ' dollars .' AS PAY
 FROM employees;
```

DISTINCT

Tüm departmanlarda çalışan olmak zorunda değil. Departmanlardan birinde hiçbir çalışan olmayabilir. Çıktıdaki tekrar eden verileri bu komut ile ortadan kaldırırız. Yani çıktıdaki tekrarları önleriz.

```
SELECT DISTINCT department_id
FROM employees;
```

KISA BİR ÖZET

1)FROM ile alanı belirle.

2)WHERE ile alanın belli bir kısmını belirtirerek alanı daralt.

3)SELECT ile o alanda arama yap.

4)ORDER BY ile de bunu bana getirirken sıralayarak getir dedik.

5) İNTRODUCTİON of FUNCTİONS

Temel SQL Sorguları İkinci Kısım

DUAL

Veri tabanında herhangi bir tablo üzerinde çalışmadığımız zaman, yazdığımız SQL cümlelerini FROM DUAL şeklinde çağırarak, istediğimiz veriyi elde edip görebiliriz.

SELECT SYSDATE FROM DUAL;

SELECT (319/29) + 12 FROM DUAL;

UPPER, LOWER ve INITCAP

Daha önce de bahsettiğim üzere veri tabanındaki veriyi çağırırken büyük küçük harf duyarlılığını bu komutlar ile kaldırabiliyoruz.

SELECT last_name

FROM employees

WHERE LOWER(last_name) = 'abel';

NOT: “abel” verisi normalde veri tabanımızda “Abel” olarak kayıtlı fakat ben burada LOWER komutu ile veriyi küçük harflerle çağırarak bu duyarlılığı kaldırmış oldum.

SELECT UPPER(last_name) from employees;

CHARACTER MANİPULATİON FUNCTİONS:

-SUBSTR: Verinin yalnızca belirttiğimiz kısmını çağırmamızı sağlar.

-LENGTH: Verinin karakter boyutunu verir.

-INSTR: Belirttiğimiz karakterin veri üzerinde kaçıncı sırada olduğunu gösterir.

-LPAD | RPAD: Veriyi solundan veya sağından belirttiğimiz karakter ile doldurur.

-TRIM: Verideki belirttiğimiz karakteri sondan veya baştan kırpmamızı sağlar.

-REPLACE: Verideki belirttiğimiz karakteri kaldırıp tercihen de yerine başka bir değer koymamızı sağlar.

SUBSTR

SELECT SUBSTR(last_name, 1, 3)
FROM employees;

NOT: Birinci karakterden başlayarak last_name verilerinin ilk üç harflerini getirir.

SELECT SUBSTR('HelloWorlddddddddddd', 6)

FROM DUAL;

NOT: Altıncı karakterden başlayarak verinin kalan tüm kısmını getirir.

LENGTH

SELECT employee_id, LENGTH(last_name) AS “Boy”

FROM employees ORDER BY Boy;

INSTR

SELECT last_name, INSTR(last_name,'a')

FROM employees;

NOT: last_name verilerinde ilk 'a' karakterinin kaçıncı sırada olduğunu gösterir.

LPAD ve RPAD

SELECT LPAD('HelloWorld', 15 ,'-')

FROM DUAL;

NOT: Yazmış olduğum verinin sol kısmına “-“ karakteri koyarak verinin karakter sayısını On beşe tamamlar.

SELECT RPAD(last_name, 10 ,'*')

FROM employees;

TRIM

SELECT TRIM(LEADING 'a' FROM 'abcba')

FROM DUAL;

NOT: Verinin başındaki “a” karakterini kaldırır ve çıktımız “bcba” olur.

SELECT TRIM(TRAILING 'a' FROM 'abcba')
FROM DUAL;

NOT: Verinin sonundaki “a” karakterini kaldırır ve çıktımız “abcb” olur.

SELECT TRIM(BOTH 'a' FROM 'abcba')

FROM DUAL;

NOT: Verinin her iki tarafındaki “a” karakterini kaldırır ve çıktımız “bcb” olur.

REPLACE

REPLACE { string1, string_to_replace, [replacement_string] }

SELECT last_name, REPLACE(last_name,'a','*') AS "Replaced"

FROM employees;

NOT: last_name verilerinde “a” karakterlerini kaldırıp yerine “*”” koyar.

SUBSTİTUTİON VARİABLES

SELECT first_name, last_name, salary, department_id

FROM employees

WHERE department_id=10;

NOT : Çağırdığımız veri statik olarak gelir.

SELECT first_name, last_name, salary, department_id

FROM employees

WHERE department_id=:bolum_kodunu_gir;

NOT : Çağırdığımız veriyi kullanıcı girer.

ROUND ve TRUNC

ROUND(45.936, 0)==ROUND(45.936)==46

ROUND(45.926, 2)==45.93

NOT: “926”daki “26” olan ilk iki hane kısmında yuvarlama işlemi yaptı.

TRUNC(45.936, 0)==TRUNC(45.936)==45

SYSDATE

SELECT last_name, hire_date, hire_date + 60 as "eklenmiş_tarih"

FROM employees;

NOT: hire_date verisinin üstüne 60 gün ekler. Fakat dikkat edilmesi gereken nokta, numerik olarak 60 eklenmez.

SELECT last_name, hire_date, (SYSDATE - hire_date)/7 AS “Kıdem”

FROM employees;

NOT: Hafta cinsinden last_name verisine sahip çalışanın kıdem süresini verir.

DATE FUNCTİONS

-MONTHS_BETWEEN: Numerik gelir

-ADD_MONTHS

-NEXT_DAY

-LAST_DAY

-ROUND

-TRUNC

SELECT last_name, hire_date, MONTHS_BETWEEN (SYSDATE, hire_date)

FROM employees;

NOT: Şu anki tarih ile hire_date verisi arasındaki tarihi numerik olarak ay cinsinden getirir.

SELECT last_name, hire_date, MONTHS_BETWEEN (SYSDATE, hire_date)

FROM employees;

WHERE MONTHS_BETWEEN

(SYSDATE, hire_date) > 340;

NOT: Şu anki tarih ile hire_date verisi arasında 340 aydan daha fazla olan verileri getirir.

SELECT ADD_MONTHS (SYSDATE, 6) as "Next Year"

FROM DUAL;

NOT: Şu anki tarihe altı ay ekleyip getirir.

SELECT NEXT_DAY (SYSDATE, 'Friday') as "Next Friday"

FROM DUAL;

NOT: Şu anki tarihten itibaren gelecek Cuma gününü tarih formatında getirir.

SELECT LAST_DAY (SYSDATE) as "End of the Month"

FROM DUAL;

NOT: Şu anki tarihte bulunduğumuz ayın son gününü getirir.

SELECT hire_date, ROUND(hire_date, 'Month') AS "Rounded HİRE_DATE"

FROM employees

WHERE department_id=50;

NOT: Diğer ayın ilk gününü date olarak yuvarlayıp getirir.

```
SELECT hire_date, ROUND(hire_date, 'Year')
FROM employees
WHERE department_id=50;
```

NOT: Burada dikkat edilmesi gereken nokta aşağı yuvarladığımızda yılın değişmediği fakat yukarı yuvarlayınca değişebildiğidir.

```
SELECT hire_date, TRUNC(hire_date, 'Month')
FROM employees
WHERE department_id=50;
```

```
SELECT hire_date, TRUNC(hire_date, 'Year')
FROM employees
WHERE department_id=50;
```

```
SELECT employee_id, hire_date,
ROUND(MONTHS_BETWEEN(SYSDATE, hire_date) AS “TENURE”
ADD_MONTHS(hire_date, 6) AS “REVİEW”
NEXT_DAY(hire_Date, 'FRİDAY'), LAST_DAY(hire_date)
FROM employees
WHERE MONTHS_BETWEEN (SYSDATE, hire_date) > 36;
```

DATA CONVERSİON

SELECT hire_date, TO_CHAR (hire_date, 'Month dd, YYYY') AS “Biçimlendirildi”

FROM employees;

SELECT hire_date, TO_CHAR (hire_date, 'fmMonth dd, YYYY')

FROM employees;

SELECT hire_date, TO_CHAR (hire_date, 'fmDAYddthsp Mon, YYYY')

FROM employees;

SELECT TO_CHAR (SYSDATE, 'hh:mm:sspm')

FROM DUAL;

SELECT TO_CHAR (4500, '99,999') FROM DUAL;

NOT: Eğer 450000 olsa “#####” çıktısı verirdi.

NOT: to_char output/çıktının formatıdır. to_number ve to_date ise inputun/girdinin formatıdır.

SELECT last_name, bonus, TO_NUMBER(bonus, '9999') As "Bonus"

FROM employees

WHERE department_id=80;

NOT: Bonus sütunu içerisinde içinde sayı olmasına rağmen tipi varchar2. O yüzden numb’a çevİrmem gerek.

SELECT TO_DATE('November 3, 2001', 'Month dd, yyyy') as "Tarihe_Cevirildi"

FROM DUAL;

SELECT TO_DATE('25 March, 2022', 'dd Month, yyyy')

FROM DUAL;

NOT: Girdiyi formatlar.

SELECT TO_CHAR(SYSDATE, 'dd Month, yyyy')

FROM DUAL;

NOT: Çıktıyı formatlar.

NOT: “fx” mutlaka bunu ara, fixle demektir.

SELECT TO_DATE ('27-Oct-95', 'DD-Mon-RR') AS "Date"

FROM DUAL;

NOT: Şu anki tarihte yıl olarak 22 deyim. Oracle buradaki 95’i olsa olsa 1995’dir diye yorumlar. 95 yerine 49 olsa 2049, 50 olsa 1950 olurdu.

NOT: Bulunduğum yıl ve aldığım değer 0-49 ise mevcut yüzyıl. Bulunduğum yıl 0-49 ve aldığım değer 50-99 ise önceki yüzyıl olarak yorumlanır.

SELECT last_name, TO_CHAR (hire_date, 'DD-Mon-YY'), TO_DATE ('01-Jan-90','DD-Mon-YY')

FROM employees

WHERE hire_date < TO_DATE ('01-Jan-90','DD-Mon-YY');

NOT: 1990’dan küçük değerleri getirmesi lazımdı fakat çıktıda büyük değerler de var.

Çünkü burada 2090’dan küçükleri kabul etti.

EKSİ NE İFADE EDER?

'-' Olduğu zaman tam sayı kısmına bakılır. -1 ise sondan bi basamağı yuvarla demek olur.

```
SELECT ROUND(30695.348,-1) as "R-1" FROM DUAL;  --30700

SELECT ROUND(30695.999,-2) as "R-2" FROM DUAL;  --30700

SELECT ROUND(30645.999,-1) as "R-1" FROM DUAL;  --30650

SELECT ROUND(30650.999,-2) as "R-2" FROM DUAL;  --30700

SELECT ROUND(30649.999,-2) as "R-2" FROM DUAL;  --30600
```

Date Conversion to Character Data

- The tables show the different format models that can be used.
- When specifying time elements, note that hours (HH), minutes (MI), seconds (SS), and AM or PM can also be formatted.

YYYY	Full year in numbers
YEAR	Year spelled out
MM	Two-digit value for month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month
DDspth	FOURTEENTH
Ddspth	Fourteenth
ddspth	fourteenth
DDD or DD or D	Day of year, month or week
HH24:MI:SS AM	15:45:32 PM
DD "of" MONTH	12 of October

Number Conversion to Character Data (VARCHAR2)

- The table illustrates some of the format elements available to use with TO_CHAR functions.

```
SELECT TO_CHAR(salary,
 '$99,999') AS "Salary"
FROM employees;
```

Salary
$24,000
$17,000

ELEMENT	DESCRIPTION	EXAMPLE	RESULT
9	Numeric position (# of 9's determine width)	999999	1234
0	Display leading zeros	099999	001234
$	Floating dollar sign	$999999	$1234
L	Floating local currency symbol	L999999	FF1234
.	Decimal point in position specified	999999.99	1234.00
,	Comma in position specified	999,999	1,234
MI	Minus signs to right (negative values)	999999MI	1234-
PR	Parenthesize negative numbers	999999PR	<1234>
EEEE	Scientific notation (must have four EEEE)	99.999EEEE	1.23E+03
V	Multiply by 10 n times (n= number of 9's after V)	9999V99	9999V99
B	Display zero values as blank, not 0	B9999.99	1234.00

Function	Description
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Date of the next occurance of day of the week specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date