応用情報技術者試験のためのSQL学習ガイド - 完全解説版

第1章：データベースとSQLの基礎知識

1.1 データベース管理システム（DBMS）の概念

データベース管理システム（Database Management System: DBMS）は、データを効率的に管理するためのソフトウェアシステムです。主な機能には以下のようなものがあります：

1. データの格納と管理

- 大量のデータを効率的に保存

- データの一貫性を保持

- データの重複を防止

2. データの保護

- アクセス制御による機密性の確保

- バックアップと復旧機能

- 同時アクセスの制御

3. データの操作

- 効率的なデータの検索

- データの更新と削除

- 複数のユーザーによる同時アクセスの管理

1.2 リレーショナルデータベースの基本概念

リレーショナルデータベースでは、データを表（テーブル）形式で管理します。以下の要素から構成されています：

1. テーブル（表）

- 行（レコード）：1件分のデータ

- 列（カラム）：データの項目

- 主キー：各行を一意に識別する値

- 外部キー：他のテーブルとの関連付けに使用する値

例えば、以下のような「社員」テーブルを考えてみましょう：

| 社員ID | 氏名 | 部署ID | 入社日 | 給与 |

|--------|------|---------|--------|------|

| 1001 | 山田太郎 | 10 | 2020-04-01 | 280000 |

| 1002 | 鈴木花子 | 20 | 2021-04-01 | 250000 |

この例では：

- 「社員ID」「氏名」「部署ID」「入社日」「給与」が列（カラム）

- 各行が1人の社員のデータ

- 「社員ID」が主キー

- 「部署ID」が外部キー（部署テーブルを参照）

第2章：SQL言語の基本構造と詳細な構文解説

2.1 SQL文の基本構造

SQLの基本構造は以下の要素から成り立っています：

1. \*\*命令部\*\*：実行する操作を指定

- SELECT, INSERT, UPDATE, DELETE など

2. \*\*対象部\*\*：操作の対象となるテーブルやカラムを指定

- FROM句によるテーブルの指定

- テーブル名、カラム名の指定

3. \*\*条件部\*\*：データの抽出や更新の条件を指定

- WHERE句による条件指定

- GROUP BY句によるグループ化

- HAVING句による集計結果の絞り込み

- ORDER BY句による並べ替え

2.2 データ定義言語（DDL）の詳細構文

2.2.1 CREATE文（テーブルの作成）

基本構文：

```sql

CREATE TABLE テーブル名 (

カラム名1 データ型1 [制約1],

カラム名2 データ型2 [制約2],

...

[テーブル制約]

);

```

具体例と詳細な解説：

```sql

CREATE TABLE 社員 (

社員ID INTEGER PRIMARY KEY, -- 主キー制約

氏名 VARCHAR(50) NOT NULL, -- NULL値を許可しない

部署ID INTEGER REFERENCES 部署(部署ID), -- 外部キー制約

入社日 DATE DEFAULT CURRENT\_DATE, -- デフォルト値の設定

給与 INTEGER CHECK (給与 >= 180000), -- チェック制約

電話番号 VARCHAR(15) UNIQUE, -- 一意性制約

CONSTRAINT 社員\_PK PRIMARY KEY (社員ID), -- テーブル制約として主キーを定義

CONSTRAINT 社員\_部署FK FOREIGN KEY (部署ID) -- テーブル制約として外部キーを定義

REFERENCES 部署(部署ID)

ON DELETE CASCADE -- 参照先が削除された場合の動作

ON UPDATE CASCADE -- 参照先が更新された場合の動作

);

```

制約の詳細説明：

1. PRIMARY KEY制約

- テーブル内で一意の値を持つ

- NULL値を許可しない

- テーブルには1つの主キーのみ定義可能

2. FOREIGN KEY制約

- 他のテーブルを参照する

- 参照整合性を保証

- 参照先の値が存在しない値は設定不可

3. CHECK制約

- カラムの値が特定の条件を満たすことを確認

- 複数の条件を論理演算子（AND, OR）で結合可能

4. DEFAULT制約

- 値が指定されない場合のデフォルト値を設定

- 関数や定数を使用可能

2.2.2 ALTER文（テーブルの変更）

基本構文：

```sql

-- カラムの追加

ALTER TABLE テーブル名

ADD カラム名 データ型 [制約];

-- カラムの変更

ALTER TABLE テーブル名

ALTER COLUMN カラム名 データ型 [制約];

-- カラムの削除

ALTER TABLE テーブル名

DROP COLUMN カラム名;

-- 制約の追加

ALTER TABLE テーブル名

ADD CONSTRAINT 制約名 制約定義;

```

具体例：

```sql

-- カラムの追加

ALTER TABLE 社員

ADD 役職 VARCHAR(20);

-- データ型の変更

ALTER TABLE 社員

ALTER COLUMN 電話番号 VARCHAR(20);

-- 制約の追加

ALTER TABLE 社員

ADD CONSTRAINT 給与制約 CHECK (給与 <= 1000000);

-- 外部キーの追加

ALTER TABLE 社員

ADD CONSTRAINT 部署FK

FOREIGN KEY (部署ID) REFERENCES 部署(部署ID);

```

2.3 データ操作言語（DML）の詳細構文

2.3.1 SELECT文（データの検索）

基本構文：

```sql

SELECT [DISTINCT] カラム名1, カラム名2, ...

FROM テーブル名

[WHERE 条件]

[GROUP BY グループ化カラム]

[HAVING 集計条件]

[ORDER BY ソートカラム [ASC|DESC]]

[LIMIT 取得件数];

```

構文要素の詳細説明：

1. SELECT句

- 取得するカラムを指定

- `\*`ですべてのカラムを選択

- `DISTINCT`で重複を除外

- カラムに別名を付けることが可能（AS句）

2. WHERE句

- 条件に合致する行を選択

- 比較演算子：=, <>, >, <, >=, <=

- 論理演算子：AND, OR, NOT

- 特殊演算子：BETWEEN, IN, LIKE, IS NULL

3. GROUP BY句

- 指定したカラムでグループ化

- 集計関数と組み合わせて使用

4. HAVING句

- グループ化した結果に対する条件指定

- 集計関数を条件に使用可能

5. ORDER BY句

- 結果の並べ替え

- ASC（昇順）またはDESC（降順）を指定可能

具体例と解説：

```sql

-- 基本的な検索

SELECT 社員ID, 氏名, 給与

FROM 社員

WHERE 給与 >= 250000;

-- グループ化と集計

SELECT

部署ID,

COUNT(\*) AS 社員数,

AVG(給与) AS 平均給与,

MAX(給与) AS 最高給与

FROM 社員

GROUP BY 部署ID

HAVING AVG(給与) >= 280000

ORDER BY 平均給与 DESC;

-- 複数テーブルの結合

SELECT

社員.氏名,

部署.部署名,

社員.給与

FROM 社員

INNER JOIN 部署

ON 社員.部署ID = 部署.部署ID

WHERE 社員.給与 >= 250000

ORDER BY 社員.給与 DESC;

```

2.3.2 INSERT文（データの追加）

基本構文：

```sql

-- 1行追加

INSERT INTO テーブル名 (カラム1, カラム2, ...)

VALUES (値1, 値2, ...);

-- 複数行追加

INSERT INTO テーブル名 (カラム1, カラム2, ...)

VALUES

(値1A, 値2A, ...),

(値1B, 値2B, ...);

-- SELECT文の結果を追加

INSERT INTO テーブル名 (カラム1, カラム2, ...)

SELECT カラム1, カラム2, ...

FROM 別テーブル

WHERE 条件;

```

具体例：

```sql

-- 1行のデータを追加

INSERT INTO 社員 (社員ID, 氏名, 部署ID, 給与)

VALUES (1003, '田中一郎', 10, 270000);

-- 複数行のデータを追加

INSERT INTO 社員 (社員ID, 氏名, 部署ID, 給与)

VALUES

(1004, '佐藤次郎', 20, 260000),

(1005, '高橋三郎', 30, 250000);

-- 別テーブルからデータをコピーして追加

INSERT INTO 社員履歴 (社員ID, 氏名, 部署ID, 給与, 記録日)

SELECT 社員ID, 氏名, 部署ID, 給与, CURRENT\_DATE

FROM 社員

WHERE 部署ID = 10;

```

2.3.3 UPDATE文（データの更新）

基本構文：

```sql

UPDATE テーブル名

SET

カラム1 = 値1,

カラム2 = 値2,

...

[WHERE 条件];

```

具体例：

```sql

-- 特定の社員の給与を更新

UPDATE 社員

SET 給与 = 給与 \* 1.1

WHERE 社員ID = 1001;

-- 部署全体の給与を更新

UPDATE 社員

SET

給与 = 給与 \* 1.05,

更新日 = CURRENT\_DATE

WHERE 部署ID = 10;

-- サブクエリを使用した更新

UPDATE 社員

SET 給与 = (

SELECT AVG(給与)

FROM 社員

WHERE 部署ID = 10

)

WHERE 部署ID = 20;

2.3.4 DELETE文（データの削除）

基本構文：

```sql

DELETE FROM テーブル名

[WHERE 条件];

```

具体例：

```sql

-- 特定の社員のデータを削除

DELETE FROM 社員

WHERE 社員ID = 1001;

-- 条件に合致する複数のデータを削除

DELETE FROM 社員

WHERE 部署ID = 10 AND 給与 < 200000;

-- サブクエリを使用した削除

DELETE FROM 社員

WHERE 部署ID IN (

SELECT 部署ID

FROM 部署

WHERE 部署名 LIKE '営業%'

);

```

2.4 高度な検索技術

2.4.1 結合（JOIN）の詳細

1. INNER JOIN（内部結合）

```sql

-- 基本構文

SELECT カラム1, カラム2, ...

FROM テーブル1

INNER JOIN テーブル2

ON 結合条件;

-- 具体例

SELECT

社員.氏名,

部署.部署名,

社員.給与

FROM 社員

INNER JOIN 部署

ON 社員.部署ID = 部署.部署ID;

```

2. LEFT OUTER JOIN（左外部結合）

```sql

-- 基本構文

SELECT カラム1, カラム2, ...

FROM テーブル1

LEFT OUTER JOIN テーブル2

ON 結合条件;

-- 具体例（部署に所属していない社員も含めて表示）

SELECT

社員.氏名,

部署.部署名

FROM 社員

LEFT OUTER JOIN 部署

ON 社員.部署ID = 部署.部署ID;

```

3. RIGHT OUTER JOIN（右外部結合）

```sql

-- 基本構文

SELECT カラム1, カラム2, ...

FROM テーブル1

RIGHT OUTER JOIN テーブル2

ON 結合条件;

-- 具体例（社員が所属していない部署も含めて表示）

SELECT

社員.氏名,

部署.部署名

FROM 社員

RIGHT OUTER JOIN 部署

ON 社員.部署ID = 部署.部署ID;

```

4. FULL OUTER JOIN（完全外部結合）

```sql

-- 基本構文

SELECT カラム1, カラム2, ...

FROM テーブル1

FULL OUTER JOIN テーブル2

ON 結合条件;

-- 具体例（両方のテーブルの全データを表示）

SELECT

社員.氏名,

部署.部署名

FROM 社員

FULL OUTER JOIN 部署

ON 社員.部署ID = 部署.部署ID;

```

2.4.2 サブクエリの活用

1. WHERE句でのサブクエリ

```sql

-- 平均給与より高い給与をもらっている社員を検索

SELECT 社員ID, 氏名, 給与

FROM 社員

WHERE 給与 > (

SELECT AVG(給与)

FROM 社員

);

-- 特定の部署の社員を検索

SELECT 社員ID, 氏名

FROM 社員

WHERE 部署ID IN (

SELECT 部署ID

FROM 部署

WHERE 部署名 LIKE '営業%'

);

```

2. FROM句でのサブクエリ（派生テーブル）

```sql

-- 部署ごとの平均給与を計算し、その結果を使用

SELECT

d.部署名,

s.平均給与

FROM (

SELECT 部署ID, AVG(給与) AS 平均給与

FROM 社員

GROUP BY 部署ID

) s

JOIN 部署 d ON s.部署ID = d.部署ID;

```

3. SELECT句でのサブクエリ（スカラーサブクエリ）

```sql

-- 各社員の給与と全社平均との差額を計算

SELECT

社員ID,

氏名,

給与,

給与 - (SELECT AVG(給与) FROM 社員) AS 平均との差額

FROM 社員;

```

2.5 高度な集計機能

2.5.1 GROUP BY句とHAVING句の詳細

```sql

-- 部署ごとの統計情報を計算

SELECT

部署ID,

COUNT(\*) AS 社員数,

AVG(給与) AS 平均給与,

MAX(給与) AS 最高給与,

MIN(給与) AS 最低給与,

SUM(給与) AS 給与総額

FROM 社員

GROUP BY 部署ID

HAVING COUNT(\*) >= 5 -- 5人以上の部署のみ表示

AND AVG(給与) > 250000; -- 平均給与が25万円を超える部署

```

2.5.2 WINDOW関数（分析関数）

```sql

-- 基本構文

SELECT

カラム1,

WINDOW関数 OVER (

[PARTITION BY 分割カラム]

[ORDER BY ソートカラム]

[フレーム句]

)

FROM テーブル名;

-- 具体例：部署ごとの給与ランキング

SELECT

社員ID,

氏名,

部署ID,

給与,

RANK() OVER (

PARTITION BY 部署ID

ORDER BY 給与 DESC

) AS 部署内給与順位

FROM 社員;

-- 累計値の計算

SELECT

社員ID,

氏名,

給与,

SUM(給与) OVER (

ORDER BY 社員ID

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

) AS 累計給与

FROM 社員;

```

2.6 トランザクション制御

2.6.1 トランザクションの基本

```sql

-- トランザクションの開始

BEGIN TRANSACTION;

-- データの更新

UPDATE 口座

SET 残高 = 残高 - 10000

WHERE 口座番号 = 'A001';

UPDATE 口座

SET 残高 = 残高 + 10000

WHERE 口座番号 = 'B001';

-- 正常終了の場合

COMMIT;

-- 異常時のロールバック

ROLLBACK;

```

2.6.2 トランザクション分離レベル

1. READ UNCOMMITTED

- 他のトランザクションの未確定データが読める

- ダーティリードが発生する可能性あり

2. READ COMMITTED

- 他のトランザクションの確定データのみ読める

- ノンリピータブルリードが発生する可能性あり

3. REPEATABLE READ

- トランザクション中は他のトランザクションによる更新の影響を受けない

- ファントムリードが発生する可能性あり

4. SERIALIZABLE

- 最も厳格な分離レベル

- 完全な直列化を保証

第3章：応用情報技術者試験対策

3.1 試験でよく出題される分野

1. データベース設計

- ER図の理解

- 正規化理論

- 参照整合性制約

2. SQL構文

- 基本的なCRUD操作

- 複雑な結合とサブクエリ

- 集計関数と集合演算

3. トランザクション管理

- ACID特性

- 同時実行制御

- ロックと分離レベル

3.2 実践的な演習問題

問題1: テーブル設計以下の要件に従ってテーブルを作成するSQLを作成してください。

要件：

- 商品マスタ（商品コード、商品名、単価、カテゴリ）

- 在庫テーブル（商品コード、倉庫コード、在庫数）

- 商品コードは商品マスタの主キー

- 在庫テーブルの主キーは商品コードと倉庫コードの組み合わせ

- 単価は1000円以上必須

- 在庫数は0以上必須

解答：

```sql

CREATE TABLE 商品マスタ (

商品コード CHAR(8) PRIMARY KEY,

商品名 VARCHAR(100) NOT NULL,

単価 INTEGER NOT NULL CHECK (単価 >= 1000),

カテゴリ VARCHAR(20)

);

CREATE TABLE 在庫 (

商品コード CHAR(8),

倉庫コード CHAR(3),

在庫数 INTEGER NOT NULL CHECK (在庫数 >= 0),

PRIMARY KEY (商品コード, 倉庫コード),

FOREIGN KEY (商品コード) REFERENCES 商品マスタ(商品コード)

);

```

問題2: データ検索以下の要件を満たすSQLを作成してください。

要件：

- カテゴリごとの商品数と平均単価を求める

- 商品数が3個以上のカテゴリのみ表示

- 平均単価が高い順に並べる

- 平均単価は小数点以下を四捨五入

解答：

```sql

SELECT

カテゴリ,

COUNT(\*) AS 商品数,

ROUND(AVG(単価)) AS 平均単価

FROM 商品マスタ

GROUP BY カテゴリ

HAVING COUNT(\*) >= 3

ORDER BY 平均単価 DESC;

```

3.3 試験合格のためのポイント

1. SQL文の実行順序を理解する

- FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY

2. 結合の種類と使い分けを理解する

- INNER JOIN

- LEFT/RIGHT OUTER JOIN

- FULL OUTER JOIN

3. サブクエリの使用方法を習得する

- WHERE句でのサブクエリ

- FROM句でのサブクエリ

- SELECT句でのサブクエリ

4. NULL値の処理に注意する

- IS NULL/IS NOT NULLの使用

- 集計関数でのNULL値の扱い

- COALESCE関数の活用

まとめ

このガイドで学んだ内容を実践的に活用するためには、以下のステップを推奨します：

1. 基本的なSQL文の作成練習

- SELECT文の基本構文

- WHERE句での条件指定

- GROUP BY句での集計

2. 複雑なクエリの理解と作成

- 複数テーブルの結合

- サブクエリの活用

- 集計関数の使用

3. 実践的な問題演習

- 過去問題の解答

- 実務を想定した課題への取り組み

- パフォーマンスを考慮したSQL作成

4. データベース設計の理解

- 正規化の概念

- ER図の作成

- 制約の適切な設定

これらの内容を十分に理解し、実践的な問題を数多く解くことで、応用情報技術者試験での高得点獲得を目指してください。